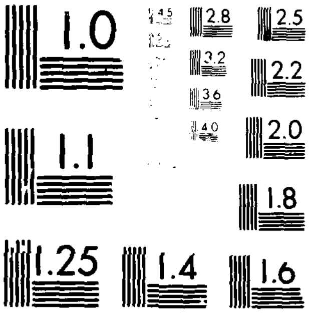
AD-A084 676 AEROSPACE CORP EL SEGUNDO CA ENGINEERING GROUP F/6 9/2
THE ECLECTIC SIMULATOR PROGRAM (ESP) USAGE GUIDE.(U)
MAY 80 E R COFFEY, H J WERTZ F04701-79-C-0080
UNCLASSIFIED TR-0080(9320)-1 SD -TR-80-21 NL

1 of 3
AD
A084676

E S P



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

REPORT SD-TR-80-21

ESP

ECLECTIC SIMULATOR PROGRAM

Usage Guide

EMMAGENE R. COFFEY
in Association with
HARVEY J. WERTZ
Mission Information Systems Division
Engineering Group
The Aerospace Corporation
El Segundo, Calif. 90245

1 May 1980

Interim Report

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

THE AEROSPACE CORPORATION

Prepared for

SPACE DIVISION
AIR FORCE SYSTEMS COMMAND
LOS ANGELES AIR FORCE STATION
P.O. Box 92960, Worldway Postal Center
Los Angeles, Calif. 90009

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18 19 **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
SD-TR-80-21

2. GOVT ACCESSION NO.
AD-A084 676

3. RECIPIENT'S CATALOG NUMBER

4. ~~TITLE (and Subtitle)~~
6 THE ECLECTIC SIMULATOR PROGRAM (ESP) USAGE GUIDE,

5. TYPE OF REPORT & PERIOD COVERED
Interim Report
7-1-75 to 12-1-79

6. ~~PERFORMING ORG. REPORT~~ NUMBER
14 TR-0080(9320)-1

7. AUTHOR(s)
Emmagene R. Coffey, Harvey J. Wertz
10

8. CONTRACT OR GRANT NUMBER(s)
15 FØ47Ø1-79-C-ØØ8Ø

9. PERFORMING ORGANIZATION NAME AND ADDRESS
The Aerospace Corporation
El Segundo, California 90245

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
12 202

11. CONTROLLING OFFICE NAME AND ADDRESS

12. REPORT DATE
11 1 May 1980

13. NUMBER OF PAGES
186

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Space Division
Air Force Systems Command
Los Angeles, Calif. 90009

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

9 Interim rept. 1 Jul 75-1 Dec 79,

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation	Hysteresis Nonlinearities
Integration	Computer Simulation
Numerical Integration	Predictor-Corrector Integration
Ordinary Differential Equations	Runge-Kutta Integration
Discontinuous Driving Functions	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Eclectic Simulator Program (ESP) is a system (a precompiler plus a collection of subroutines) that permits the fast, easy solution of ordinary differential equations. Any user with a general knowledge of FØRTRAN can utilize ESP's many labor-saving devices to code a problem with minimal effort. Special ESP features permit translation of engineering blocks, discontinuities, and hysteresis patterns directly into computer code, and the use

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS *(Continued)*

Simulation Language
Precompiler
Matrix Expressions

20. ABSTRACT *(Continued)*

of WHELP in conjunction with ESP facilitates efficient coding of matrix algebra equations. Simple input cards enable the user to directly control solution and timing accuracy and to specify or change run times, initial conditions, and various other parameters easily when making multiple or stacked runs. Finally, ESP allows the user to select from a wide variety of output options. This manual is intended to be both a learning tool for the novice and a detailed reference for the experienced user.

PREFACE

This revision of The Eclectic Simulator Program Usage Guide was prompted by a number of modifications and improvements to the ESP package which have been made since 1975. The primary reason for the changes was the desire to have common versions of ESP and WHELP available on both the CDC 7600 and the IBM 370. From the user's point of view this has been achieved even though both WHELP and PRECØMP are written in PLI for the IBM 370 and in FØRTRAN for the CDC 7600. There were, however, a number of changes required which will affect existing programs. The first of these is a revision of variable and common block names. The second is a revision and restructuring of the major subroutines to simplify program flow and logic and to improve error control and handling of potential job abort situations. The third is replacement of the former default integration algorithm with the Shampine variable-order/variable step routine, which appears to outperform the Hamming Predictor Corrector in both speed and accuracy on a variety of applications tested thus far.

In effecting these changes to the ESP package, considerable effort has been made to minimize changes required of the user. The only pervasive changes affecting the user are the shortening of variable, subroutine, and common block names to six characters or less and some rearranging of common blocks, both necessitated by IBM FØRTRAN constraints. In general, the contents of this guide reflect a CDC orientation with IBM counterparts noted where possible. However, an ESP program written according to this guide may be run interchangeably on either IBM or CDC by simply changing the appropriate control cards.

In addition to revisions of all portions of this guide reflecting the above changes, the appendix relating to WHELP (Appendix H) has been greatly expanded to include all current capabilities of WHELP. Notice that WHELP may be used with or without ESP, and thus Appendix H may be used without reference to the remainder of the guide and is in fact the most complete current documentation of WHELP.

Hopefully, this will be the final major rewrite of both the software and the documentation. In particular, this edition of the manual has been designed for easy updating when the inevitable (wishfully small) changes are made to the software.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Concluded)

FIGURES

I. INTRODUCTION

SECTION I

INTRODUCTION

The Eclectic Simulator Program (ESP) enables the user to solve ordinary differential equations with speed, accuracy, versatility, and minimal effort. The user codes only that information unique to his particular problem: the differential equations, initial conditions, and desired output. This information must be coded in ESP language, which is a special purpose programming shorthand developed just for this program, and documented in this manual. ESP then does literally all the remaining work.

To accomplish its purpose, the ESP system is composed of two parts--a precompiler and a set of FØRTRAN subroutines. The precompiler reads the ESP shorthand code written by the user (This code will not look like a FØRTRAN program.) and translates it into FØRTRAN, while adding the necessary cards (such as CØMMØN block, DIMENSIØN statements, and RETURNS) to produce complete and executable FØRTRAN subroutines. The output of this precompiler is then joined with the second part of the ESP package, the subroutines which do the integration and other chores, to form a complete FØRTRAN program which is then executed by the computer.

ESP is not only highly efficient in terms of both user's effort and computer running time, but it is also highly flexible. A number of special capabilities, labor saving devices, and alternate means to the same ends are part of the package; however, the user has considerable latitude in deciding which features he will use and how large or how small a problem he wishes to solve. Briefly listed below are some of the distinctive features and capabilities of ESP:

- The derivatives are normally defined as first-order differential equations, but may instead be translated directly from engineering block diagrams to *BLØCK cards without any intervening algebra.

- The basic integration algorithm is the highly efficient SHAMPINE[*] method, which combines variable stepsize with variable order integration in response to continuous error checks, but the user may opt to run ESP using any of several other integration algorithms, namely, second- or fourth-order Runge-Kutta, or Predictor-Corrector (which uses Runge-Kutta as a starter), any of which may be run with either a fixed or variable stepsize.

- Significant sign changes, discontinuous driving functions, hysteresis nonlinearities and the like may all be accurately and easily coded into the system by means of special ESP language command cards.

- Output from an ESP program may take many forms, such as automatically formatted print, user-formatted print, calcomp pen plots, printer plots, microfilm plots, or magnetic tape files.

- Since inputs such as initial conditions, run times, and parameters can be easily changed, a series of runs or a set of "stacked" runs can be made with a minimum of effort.

- The user can directly control the degree of accuracy required for the problem solution and also for the timing of discontinuities with simple input cards.

- Vector-Matrix expressions can be used in their natural form to compute derivatives, by using WHELP along with ESP.

This manual attempts to meet the needs of both the novice and the expert user of ESP. It is hoped that sufficient explanation and examples have been given in Sections II through VII to enable the uninitiated to write a successful program. On the other hand, considerable detail has been included throughout to aid all users in answering their own questions and debugging their own programs.

Section II includes a straightforward example of ESP usage, from problem definition through printed and plotted output. Careful study of this example and its annotation should give the user a helpful overview of how ESP works and how the various parts of user-coding relate to each other.

[*] Referred to in this manual as ADAMS because it is an Adams-like method and SHAMPINE is too long for a FØRTRAN name

Following this example, material is arranged topically by sections, one section for each major aspect of setting up an ESP program: defining the derivatives, selecting the integration method, modeling discontinuities, specifying output, and defining input. Each section begins with an overview of the capabilities relating to the section topic, and then discusses each in detail, starting with the simplest and most basic usage and progressing to more complicated options and considerations near the end.

It is strongly recommended that the user's first attempt at coding ESP involve a fairly simple problem or a simplified version of a larger problem, and that more complex aspects of ESP be added only after the basic ones are well understood and seem to be working properly. In keeping with this approach, it is suggested that the user study the example problems (Section II) and read the first few pages of each of Sections III through VII before attempting to code his first problem. Later parts of Sections III through VII and the Appendices may be regarded more as reference material and used only as needed, although particular attention should be called to Appendix D, Program Control and Execution, for those who wish to understand more fully how ESP works. The Table of Contents and Index should facilitate easy location of any other material of interest.

II. SOME SIMPLE
EXAMPLE PROBLEMS

SECTION II

SOME SIMPLE EXAMPLE PROBLEMS

To aid the user in obtaining an overview of how ESP works, this section consists solely of two example problems. The first is a very simple or minimum problem and includes the statement of the problem and the coding required to solve it using ESP. The second example is slightly more complex and is fully discussed from problem definition through analysis, coding, and resulting printout.

A. EXAMPLE 1

Problem: Integrate the following differential equations from t = 0 to t = 10.0 sec, printing t, Y, and $\dot{Y}$ every 0.5 sec:

$$\dot{Y} = \cos\theta\, Y + Bt$$

$$\ddot{Y} = \dot{Y} + \sin\theta\, Y$$

where

all initial conditions = 0.

$\theta = 0.4{*}t$

$B = 0.142$

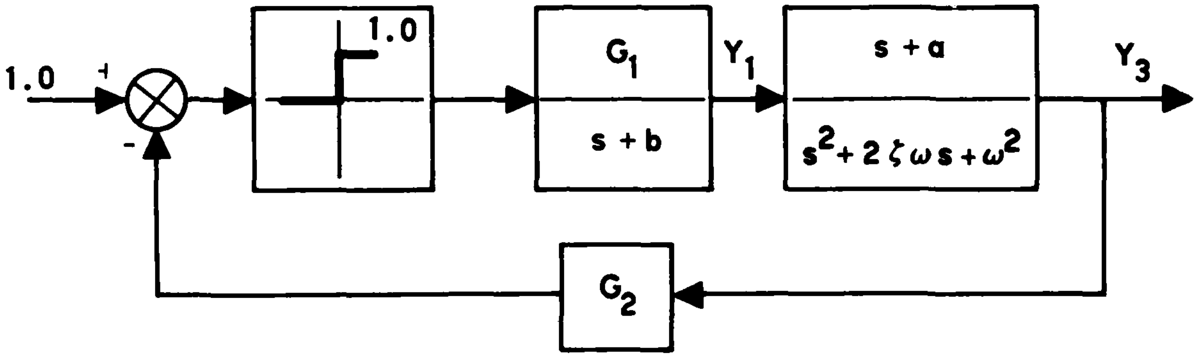
Coding:

[Control cards--see APPENDIX B]

```
*DERIVS
    THETA = 0.4 * T
    DY(1) = COS(THETA) * Y(1) + 0.142 * T
    DY(2) = DY(1) + SIN(THETA) * Y(1)
*ENDDERIVS
*PRINT TIME = T $ Y = Y(1) $ YDOT = Y(2) $ $
*RUN 2 0. 0.5 10.0 $
```

B. EXAMPLE 2

This example will be presented in detail from problem definition through to the output of a completed program in five steps. Step 1 is the statement of a problem as the typical user might define it. Step 2 shows a step-by-step analysis of this problem and translation of its characteristics into ESP code, while Step 3 illustrates the actual arrangement of this code. Steps 4 and 5 are provided by ESP and show the FØRTRAN output of the precompiler and the actual printed and plotted output requested by the user.



STEP 1: STATEMENT OF THE PROBLEM

Integrate the above system from T=0. to T=0.5

Inputs: G_1 = 5.0

G_2 = 1.0

a = 0.1

b = 0.01

ζ = 0.5

ω = 1.0E1

All initial conditions = 0.

Outputs: Print every 0.02 second the following values and labels:

Time = T
Error = $1.0 - G_2Y_3$
Output = Y_3

Plot: Output versus Time
Error versus Time

STEP 2: ANALYSIS OF THE PROBLEM

1. The number of integrations required (3), the run interval and the print interval will be specified on the *RUN card

```
*RUN 3   0.0   0.02   0.5   $
```

[See Section VII-A]

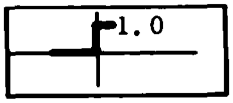
2. Input constants will be input on a *PAR card and equivalenced to their names in the equations so that later they may be easily changed.

$$\text{Set}\begin{cases}\text{PAR(1)} = G_1 = \text{G1} = 5.\\ \text{PAR(2)} = b = \text{B} = 0.01\\ \text{PAR(3)} = a = \text{A} = 0.1\\ \text{PAR(4)} = \zeta = \text{ZETA} = 0.5\\ \text{PAR(5)} = \omega = \text{ØMEGA} = 1.0\text{E}1\\ \text{PAR(6)} = G_2 = \text{G2} = 1.0\end{cases}$$

by using

```
*PAR   5.   0.01   0.1   0.5   1.E1   1.0   $
```

[See Section VII-C]

3. 

represents the following characteristics:

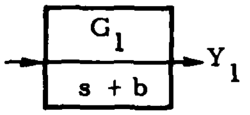
input = $1.0 - G_2 * Y_3$

$$\text{output} = \begin{cases}\text{If input} \leq 0, \text{ output} = 0.\\ \text{If input} > 0, \text{ output} = 1.0\end{cases}$$

To detect the exact point at which the value of input changes sign and to set the proper output, the *SWTCH feature will be used:

```
*SWTCH  1  1.0  $  0.  $  1.0-PAR(6)*Y(3)  $
```

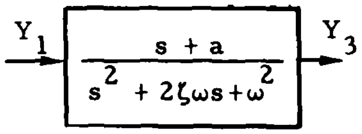
[See Section V-A]

4. 

is equivalent to $\dot{Y}_1 = G_1$ *input-b*Y(1) where input = SWCH1 which is the value resulting from the *SWTCH statement.

This will be coded as

```
        DY(1) = G1*SWCH1 - B*Y(1)            [See Section III-A]
```

5. 

is a second-order block which is equivalent to $Y_3 = Y_1[(s+a)/(s^2 + 2\zeta\omega s + \omega^2)]$

This could be solved for $\dot{Y}_3$ in terms of its auxiliary function Y_2 and coded as

```
        DY(2) = A*Y(1) - ØMEGA**2*Y(3)
        DY(3) = -2.0*ZETA*ØMEGA*Y(3) + Y(2) + Y(1)
```

but it is faster and easier to code it by using the *BLØCK input feature

```
 *BLOCK 2 1.0 A 2.*ZETA*ØMEGA ØMEGA**2 Y(3) Y(2) Y(1) $
                                             [See Section III-C]
```

6. Printing and storing of plot data will be done by using the *PRINT statement

```
  *PRINT  TIME=PLØT(1)=T  $  ERRØR=PLØT(2)=1.0-PAR(6)*Y(3)  $
          ØUTPUT=PLØT(3)=Y(3)  $  $        [See Section VI-A-1]
```

7. Printer plots with all default features will be generated by using

```
        *GRAPH 1 3
            ØUTPUT VERSUS TIME            [See Section VI-B-2]
        *GRAPH 1 2
            ERRØR VERSUS TIME
```

8. A title will be assigned to all output pages by using the *TITLE card

```
*TITLE     EXAMPLE FØR ESP MANUAL
```

[See Section VII-G-2]

```
     7600 PRECOMP .3 OCTOBER 1979.] Printed by PRECOMP
   *DERIVS
         EQUIVALENCE (G1,PAR(1)),(B,PAR(2)),(A,PAR(3)),(ZETA,PAR(4))
        * ,(OMEGA,PAR(5)),(G2,PAR(6))
A  *SWTCH 1 1.0 $ 0. $ 1.0-PAR(6)*Y(3) $] (input will be computed in SWINPT, output in DERIVS)
         DY(1) = G1 *  SWCH1  -B *Y(1)
   *BLOCK 2 1.0  A  2.*ZETA*OMEGA  OMEGA**2  Y(3)  Y(2)  Y(1)  $
   *ENDDERIVS
   *PRINT  TIME=PLOT(1)=T $ ERROR=PLOT(2)=1.0-PAR(6)*Y(3) $
B     OUTPUT=PLOT(3)=Y(3) $ $
C  *TITLE  EXAMPLE FOR ESP MANUAL

  THE CARD ABOVE AND CARDS LISTED BELOW ARE COPIED TO TAPE12 FOR LATER INPUT] Printed by PRECOMP
   *PAR 5. .01 .1 .5 1.E1  1. $
D  *RUN 3 0 0.02  0.5
   *GRAPH 1 3
E        OUTPUT VERSUS TIME
   *GRAPH 1 2
         ERROR VERUS TIME

   THE USEWLP FLAG IS =0.]
                          ] Printed by PRECOMP
  TIME =       .028 SEC. ]
```

STEP 3: CODING OF THE PROGRAM

Page 1 of actual listing, showing such items as exact card format and deck structure, provided by user. All subsequent coding is written by the ESP precompile program.

A is the derivative equations segment, which will be translated into FØRTRAN and used as the core of SUBRØUTINE DERIVS.

B is the output segment, which will be translated to form the core of SUBRØUTINE ØUTPUT.

C will produce a title on the output.

D is the "run-time" cards, provided by the user, which specify the characteristics of each run.

E is the cards which produce plots after run completion.

```
      PROGRAM MAIN       76/76    OPT=1                          FTN 4.6+452       06/07/79  13.54.09       PAGE      1

 1                 PROGRAM MAIN(TAPE11, TAPE12, INPUT=TAPE12, OUTPUT)
                   EXTERNAL DERIVS,ADAMS   ,ADMNTP
                   CALL ESPII(DERIVS,ADAMS   ,ADMNTP )
                   END



    SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
   66  MAIN

FILE NAMES        MODE
   20  INPUT                 40  OUTPUT                     0  TAPE11                 20  TAPE12

EXTERNALS          TYPE   ARGS
       ADAMS                0                          ADMNTP              0
       DERIVS               0                          ESPII               3

STATISTICS
  PROGRAM LENGTH              16B      14
  BUFFER LENGTH               60B      48
        47000B SCM USED
```

STEP 4: FØRTRAN VERSION OF THE PROGRAM

Beginning of FØRTRAN subroutines written by the precompiler from the user's coding

This main program is normally written entirely by PRECØMP. If the user writes his own, it should be placed at the beginning of the deck.

```
SUBROUTINE DERIVS      76/76    OPT=1                         FTN 4.6+452        06/07/79  13.54.09         PAGE       1

              ┌ SUBROUTINE DERIVS(T,Y,DY,STOP)
            A │ DIMENSION Y(100), DY(100), PAR(100)
              │ COMMON/SWTCHS/SWTCH(50),SWMEM(50,4),MAXSWS,MAXMEM,NEVENT
              ┴ COMMON/PARS/PAR
              │ EQUIVALENCE (G1,PAR(1)),(B,PAR(2)),(A,PAR(3)),(ZETA,PAR(4))
              │* ,(OMEGA,PAR(5)),(G2,PAR(6))
            B │ IF(SWTCH( 1).GT.0)SWCH 1=1.0 ┐Computes output of switch.
              │ IF(SWTCH( 1).LE.0)SWCH 1=0.  ┘Input is computed in SUBROUTINE SWINPT.
              │ DY(1) = G1 * SWCH1  -B *Y(1)
              │ TJKQA0 = Y(1)
              │ DY(3  ) = Y(2  ) + TJKQA0 * (1.0)-Y(3  )*(2.*ZETA*OMEGA)
              ┴ DY(2  ) = TJKQA0 * (A)-Y(3  )*(OMEGA**2)
            C ┌ RETURN
              └ END


      SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  DERIVS

VARIABLES       SN  TYPE            RELOCATION
    2  A            REAL                PARS            1  B           REAL                 PARS
    0  DY           REAL        ARRAY    F.P.           0  G1          REAL                 PARS
    5  G2           REAL                PARS          373  MAXMEM      INTEGER              SWTCHS
  372  MAXSWS       INTEGER             SWTCHS        374  NEVENT      INTEGER              SWTCHS
    4  OMEGA        REAL                PARS            0  PAR         REAL        ARRAY    PARS
    0  STOP         REAL       *UNUSED   F.P.          62  SWMEM       REAL        ARRAY    SWTCHS
    0  SWTCH        REAL        ARRAY   SWTCHS         31  SWCH1       REAL
    0  T            REAL       *UNUSED   F.P.          32  TJKQA0      REAL
    0  Y            REAL        ARRAY    F.P.           3  ZETA        REAL                 PARS

COMMON BLOCKS   LENGTH
      SWTCHS      253
      PARS        100

STATISTICS
  PROGRAM LENGTH                 33B       27
  SCM LABELED COMMON LENGTH     541B      353
         47000B SCM USED
```

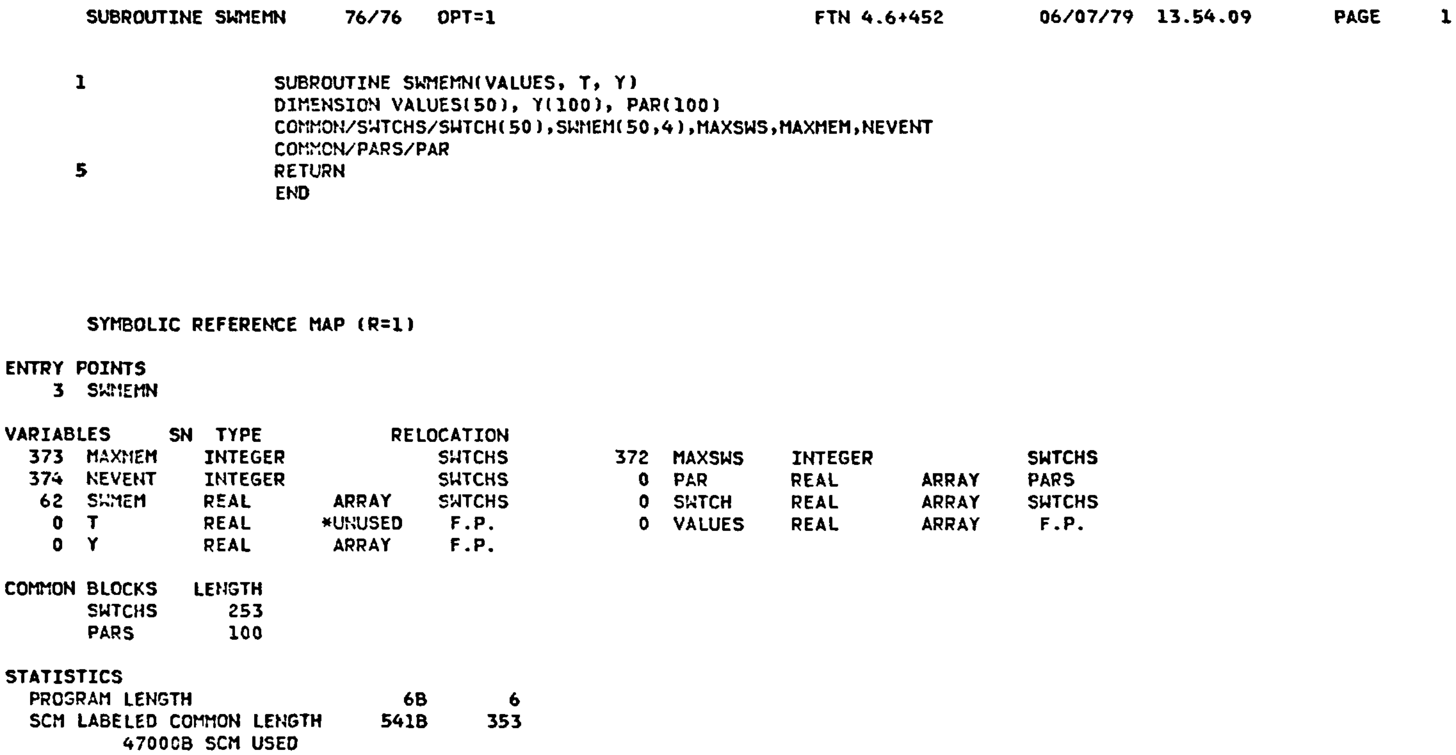
This routine contains the definitions of all derivatives that are to be integrated, and it computes the output of all switches.

All cards in sections A and C are written entirely by PRECØMP; they are identical for each ESP program. Cards in section B are the FØRTRAN equivalent of the user's derivative segment, as they have been translated by PRECØMP; and as they will be called and evaluated by the integration package.

```
SUBROUTINE SWINPT      76/76    OPT=1                          FTN 4.6+452        06/07/79  13.54.09        PAGE     1

1                   ┌SUBROUTINE SWINPT(VALUES,T,Y)
                    │DIMENSION VALUES(50), Y(100), PAR(100)
                  A │COMMON/SWTCHS/SWTCH(50),SWMEM(50,4),MAXSWS,MAXMEM,NEVENT
                    └COMMON/PARS/PAR
5                 B └VALUES( 1)=1.0-PAR(6)*Y(3)
                  C ┌RETURN
                    └END


     SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  SWINPT

VARIABLES      SN  TYPE            RELOCATION
  373  MAXMEM     INTEGER                SWTCHS           372  MAXSWS     INTEGER                 SWTCHS
  374  NEVENT     INTEGER                SWTCHS             0  PAR        REAL        ARRAY       PARS
   62  SWMEM      REAL         ARRAY     SWTCHS             0  SWTCH      REAL        ARRAY       SWTCHS
    0  T          REAL        *UNUSED     F.P.              0  VALUES     REAL        ARRAY        F.P.
    0  Y          REAL         ARRAY      F.P.

COMMON BLOCKS    LENGTH
       SWTCHS       253
       PARS         100

STATISTICS
  PROGRAM LENGTH                     12B       10
  SCM LABELED COMMON LENGTH         541B      353
           47000B SCM USED
```

*This routine defines the inputs to any *SWTCHs used.*

*Sections A and C are written entirely by PRECØMP; Section B defines the inputs to any switches that have been coded by the user on *SWTCH cards. The output of these switches is computed in SUBRØUTINE DERIVS.*



```
SUBROUTINE SWMEMN     76/76   OPT=1                           FTN 4.6+452        06/07/79  13.54.09       PAGE      1

                      SUBROUTINE SWMEMN(VALUES, T, Y)
                      DIMENSION VALUES(50), Y(100), PAR(100)
                      COMMON/SWTCHS/SWTCH(50),SWMEM(50,4),MAXSWS,MAXMEM,NEVENT
                      COMMON/PARS/PAR
                      RETURN
                      END


SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  SWMEMN

VARIABLES      SN  TYPE            RELOCATION
  373  MAXMEM      INTEGER               SWTCHS          372  MAXSWS     INTEGER                SWTCHS
  374  NEVENT      INTEGER               SWTCHS            0  PAR        REAL        ARRAY      PARS
   62  SWMEM       REAL         ARRAY    SWTCHS            0  SWTCH      REAL        ARRAY      SWTCHS
    0  T           REAL        *UNUSED    F.P.             0  VALUES     REAL        ARRAY       F.P.
    0  Y           REAL         ARRAY     F.P.

COMMON BLOCKS   LENGTH
       SWTCHS      253
       PARS        100

STATISTICS
  PROGRAM LENGTH                  6B       6
  SCM LABELED COMMON LENGTH     541B     353
         47000B SCM USED
```

This routine defines the inputs to any *SWMEMs used. Since none are used in this example, the routine is a dummy or do-nothing routine, written entirely by PRECØMP.

```
SUBROUTINE OUTPUT     76/76   OPT=1                         FTN 4.6+452      06/07/79  13.54.09       PAGE     1

                 ┌SUBROUTINE OUTPUT(T,Y,DY,PLOT,PRINT,STOP)
               A │DIMENSION Y(100), PAR(100), PLOT(100), PRINT(60), DY(100)
                 │CCMMON/SWTCHS/SWTCH(50),SWMEM(50,4),MAXSWS,MAXMEM,NEVENT
                 └COMMON/PARS/PAR
                 ┌PRINT( 1)=PLOT(1)=T
               B │PRINT( 2)=PLOT(2)=1.0-PAR(6)*Y(3)
                 └PRINT( 3)=PLOT(3)=Y(3)
               C ┌RETURN
                 └END


SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  OUTPUT

VARIABLES      SN  TYPE            RELOCATION
    0  DY          REAL       ARRAY     F.P.          373  MAXMEM     INTEGER               SWTCHS
  372  MAXSWS      INTEGER              SWTCHS        374  NEVENT     INTEGER               SWTCHS
    0  PAR         REAL       ARRAY     PARS            0  PLOT       REAL        ARRAY      F.P.
    0  PRINT       REAL       ARRAY     F.P.            0  STOP       REAL       *UNUSED     F.P.
   62  SWMEM       REAL       ARRAY     SWTCHS          0  SWTCH      REAL        ARRAY     SWTCHS
    0  T           REAL                 F.P.            0  Y          REAL        ARRAY      F.P.

COMMON BLOCKS   LENGTH
       SWTCHS      253
       PARS        100

STATISTICS
  PROGRAM LENGTH                    16B       14
  SCM LABELED COMMON LENGTH        541B      353
         47000B SCM USED
```

This routine defines the values to be printed and stored for plotting.

Sections A and C are written by PRECØMP.

B contains the FØRTRAN version of the user's print and plot storage instructions.

```
      SUBROUTINE ICCOMP     76/76   OPT=1                          FTN 4.6+452        06/07/79  13.54.09       PAGE      1

    1                SUBROUTINE ICCOMP (T,Y)
                     DIMENSION Y(100),PAR(100)
                     RETURN
                     END


      SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  ICCOMP

VARIABLES     SN  TYPE           RELOCATION
    6  PAR        REAL      *UNDEF                     0  T          REAL      *UNUSED    F.P.
    0  Y          REAL       ARRAY     F.P.

STATISTICS
  PROGRAM LENGTH                  152B     106
          47000B SCM USED
```

*This routine is called once at the beginning of program execution to compute any needed initial conditions or values. Any user coding included between *ICCØMP and *ENDØUT will be written as part of this routine. Since no initial computations were needed for this problem, this is merely a dummy or do-nothing routine and was written entirely by PRECØMP.*

```
        S C O P E   L O A D   M A P                   LOADER VERSION 1.0   06/07/79  13.54.35.  PAGE   1

PROGRAM  WILL BE ENTERED AT  MAIN      (   176)          SCM LENGTH     55501          LCM LENGTH       0

        BLOCK         ADDRESS    LENGTH      FILE

        MAIN            110         76        LGO
         /SWTCHS/       206        375     UL-LIB1
         /PARS/         603        144     UL-LIB1
        DERIVS          747         33        LGO      routines written by PRECOMP from user's code
        SWINPT         1002         12        LGO
        SWMEMN         1014          6        LGO
        OUTPUT         1022         16        LGO
        ICCOMP         1040        152        LGO
         /READIN/      1212        125     UL-LIB1
        READIT         1337        165     UL-LIB1
        PACKER         1524        121     UL-LIB1
        DECOD          1645        314     UL-LIB1
        NEXTCHR        2161         36     UL-LIB1
         /REFRO/       2217          1     UL-LIB1
        FILLEUF        2220         31     UL-LIB1
        SKIPFIL        2251         22     UL-LIB1
        ICKBLNK        2273         35     UL-LIB1
        TIMEIN         2330         41     UL-LIB1
         /BLANK/       2371       7640     UL-LIB1
         /UNIP1/      12231        150     UL-LIB1
         /UNIP2/      12401        211     UL-LIB1
         /HMAXMN/     12612          4     UL-LIB1
         /MISCEL/     12616       1051     UL-LIB1
         /STFCON/     13667          5     UL-LIB1
         /BASIC/      13674        457     UL-LIB1
         /RKCONT/     14353          1     UL-LIB1
         /SWHPAR/     14354       1052     UL-LIB1
         /SWDBUG/     15426          1     UL-LIB1
         /NDISPR/     15427          1     UL-LIB1      routines and common blocks supplied by ESP in
        ESPII         15430       3240     UL-LIB1      2NEWRESP file.
        ESPRNT        20670        114     UL-LIB1
        ESPLOT        21004         67     UL-LIB1
         /STFPAR/     21073          4     UL-LIB1
        ESPCTL        21077       1227     UL-LIB1
        SECNZR        22326        522     UL-LIB1
        SWINIT        23050        275     UL-LIB1
        SWTCHE        23345        574     UL-LIB1
        ADAMS         24141       1176     UL-LIB1
        ADMNTP        25337        204     UL-LIB1
        RESTOR        25543         72     UL-LIB1
        SCALEPR       25635        126     UL-LIB1
         /PLOTSYM/    25763          1     UL-LIB1
         /GRAFHP/     25764         62     UL-LIB1
        GRAFH2        26046       1066     UL-LIB1
        GRAFHX        27134       5656     UL-LIB1
        GRAFH         35012        165     UL-LIB1
        ENCOD         35177        165     UL-LIB1
        IDECOD        35364         57     UL-LIB1
        JUNK          35443          7     UL-LIB1
```

```
S C O P E   L O A D   M A P                        LOADER VERSION 1.0    06/07/79  13.54.35.  PAGE   2

EVENTS         35452          6    UL-LIB1
SYSTEM=        35460         14    UL-LIB1
NUMBER         35474        236    UL-LIB2
 /CALCOM/      35732         23    UL-LIB2
NUPLOT         35755        712    UL-LIB2
PLOTS          36667        220    UL-LIB2
FRAMES         37107         13    UL-LIB2
FRAMXX         37122        131    UL-LIB2
LINGRD         37253        235    UL-LIB2
LOGGRD         37510        257    UL-LIB2
STDGRD         37767       1201    UL-LIB2
LEVEL1         41170        441    UL-LIB2
LEVEL2         41631        662    UL-LIB2
 /PINOUC/      42513         16    UL-LIB2
PARRAY         42531        434    UL-LIB2
PINOUT         43165        311    UL-LIB2
PLTSYM         43476        362    UL-LIB2
SYMBOL         44060        247    UL-LIB2
IDFRAM         44327        153    UL-LIB2
 /PLBUFF/      44502        765    UL-LIB2
BUFF           45467        362    UL-LIB2
GENGRD         46051        243    UL-LIB2
LEVEL          46314         74    UL-LIB2
NEWGRD         46410        336    UL-LIB2
 /STP.END/     46746          1    SL-FORTX
 /FCL.C./      46747         23    SL-FORTX
 /Q8.IO./      46772        136    SL-FORTX
Q8NTRY=        47130          1    SL-FORTX
COMIO=         47131         60    SL-FORTX
ENCODE=        47211        144    SL-FORTX
ENDFIL=        47355         45    SL-FORTX
EOF            47422         20    SL-FORTX
FECMSK=        47442         41    SL-FORTX
FLTIN=         47503        156    SL-FORTX
FLTOUT=        47661        315    SL-FORTX
FMTAP=         50176        373    SL-FORTX
FORSYS=        50571        533    SL-FORTX
FORUTL=        51324         44    SL-FORTX
GETFIT=        51370         43    SL-FORTX
INCOM=         51433        262    SL-FORTX
 /IO.BUF./     51715        227    SL-FORTX
INPB=          52144        324    SL-FORTX
INPC=          52470        173    SL-FORTX
KODER=         52663        467    SL-FORTX
KRAKER=        53352        435    SL-FORTX
OUTB=          54007        215    SL-FORTX
OUTC=          54224        174    SL-FORTX
OUTCOM=        54420        204    SL-FORTX
REWIND=        54624         36    SL-FORTX
SYSTEM         54662         30    SL-FORTX
CLOCK=         54712         55    SL-FORTX
GOTOER=        54767         14    SL-FORTX
REMARK         55003          3    SL-FORTX
ALOG           55006         77    SL-FORTX
```

routines used by ESP for plotting: loaded from 3FTNPLOTLIB

System routines

```
       S C O P E   L O A D   M A P                    LOADER VERSION 1.0   06/07/79  13.54.35.  PAGE   3

       EXP          55105        100    SL-FORTX
       ITOJ=        55205         16    SL-FORTX
       SINCOS=      55223         74    SL-FORTX
       SQRT         55317         46    SL-FORTX
       SYSAID=      55365          1    SL-FORTX
       SYS=1ST      55366         62    SL-FORTX
       XTOI=        55450         10    SL-FORTX
       XTOY=        55460          7    SL-FORTX
       NUMARG       55467         12    SL-FORTX
A   *SWITCHES  1
    *SWMEMCNT  0
    *HEADINGS  TIME          ERROR         OUTPUT        $
B   *TITLE  EXAMPLE FOR ESP MANUAL
    *PAR 5. .01 .1 .5 1.E1  1. $
    *RUN 3 0 0.02  0.5
```

System routines (cont)

Printed by ESP: cards in A are written by PRECOMP,
cards in B by user.

```
                                   ESP-II RUN-TIME SYSTEM     09 AUGUST 1977


06/07/79                  EXAMPLE FOR ESP MANUAL

                                    SOLUTION BEGINS AT      0.
                                    PRINTOUT INTERVAL  =    2.000000000E-02
                                    SOLUTION ENDS AT        5.000000000E-01
                                    DECIMALS PRINTED   =     6


                                    EPS VALUE = 1.0000000E-06


                                    ALL Q VALUES    =  1.0000000E-10


                                    ALL HSW VALUES =  1.0000000E-06


                                    ALL IV VALUES ARE ZERO



                                    NONZERO PAR VALUES
PAR(  1) = 5.0000000E+00  PAR(  2) = 1.0000000E-02  PAR(  3) = 1.0000000E-01  PAR(  4) = 5.0000000E-01
PAR(  5) = 1.0000000E+01  PAR(  6) = 1.0000000E+00
```

Data printed by the user from ICCOMP will be positioned here.

STEP 5: OUTPUT OF THE PROGRAM

Beginning of job output; all data on this page is printed automatically for every run by ESP. If any initial values are nonzero, they will be printed in the same format as the PARS.

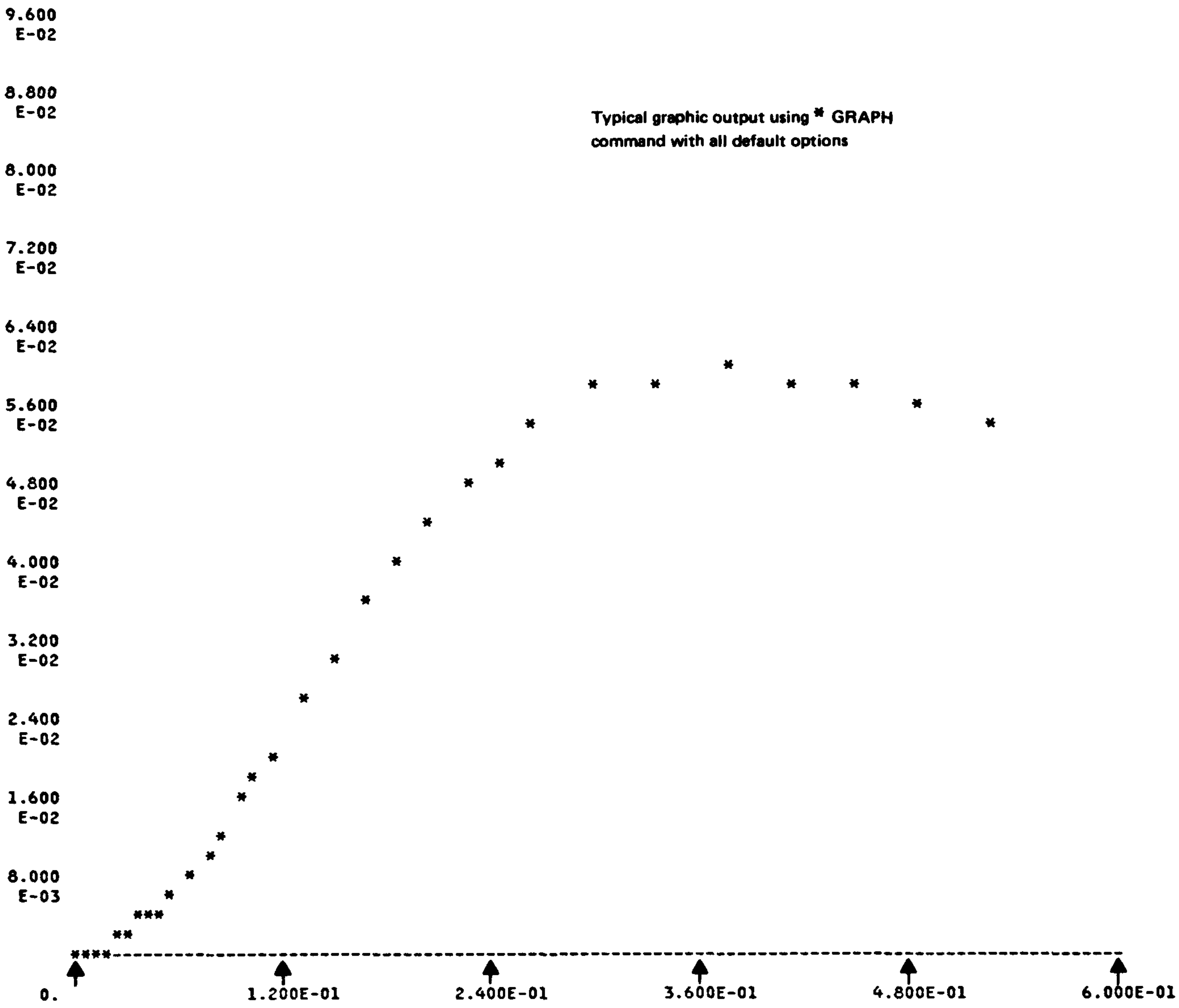
```
06/07/79      EXAMPLE FOR ESP MANUAL                                                         PAGE NO.   1

   TIME              ERROR             OUTPUT
   0.                1.000000E+00      0.
   2.000000E-02      9.990660E-01      9.340322E-04
   4.000000E-02      9.965250E-01      3.474972E-03
   6.000000E-02      9.927570E-01      7.242974E-03
   8.000000E-02      9.881173E-01      1.188271E-02
   1.000000E-01      9.829282E-01      1.707176E-02
   1.200000E-01      9.774737E-01      2.252631E-02
   1.400000E-01      9.719956E-01      2.800439E-02
   1.600000E-01      9.666930E-01      3.330703E-02
   1.800000E-01      9.617223E-01      3.827769E-02
   2.000000E-01      9.571998E-01      4.280018E-02
   2.200000E-01      9.532044E-01      4.679557E-02
   2.400000E-01      9.497817E-01      5.021830E-02
   2.600000E-01      9.469483E-01      5.305174E-02
   2.800000E-01      9.446964E-01      5.530356E-02
   3.000000E-01      9.429989E-01      5.700108E-02
   3.200000E-01      9.418133E-01      5.818668E-02
   3.400000E-01      9.410865E-01      5.891355E-02
   3.600000E-01      9.407582E-01      5.924184E-02
   3.800000E-01      9.407649E-01      5.923514E-02
   4.000000E-01      9.410423E-01      5.895771E-02
   4.200000E-01      9.415282E-01      5.847185E-02
   4.400000E-01      9.421638E-01      5.783616E-02
   4.600000E-01      9.428960E-01      5.710399E-02
   4.800000E-01      9.436775E-01      5.632254E-02
   5.000000E-01      9.444678E-01      5.553221E-02
 *GRAPH 1 3

TIME =       .027 SEC.

MAXSTEP   3.7515E-02    MINSTEP   1.1180E-09    NUMSTP     58    PLOT PTS.   59    NTAP11      59
```

Output specified by user; format is automatically controlled by ESP.

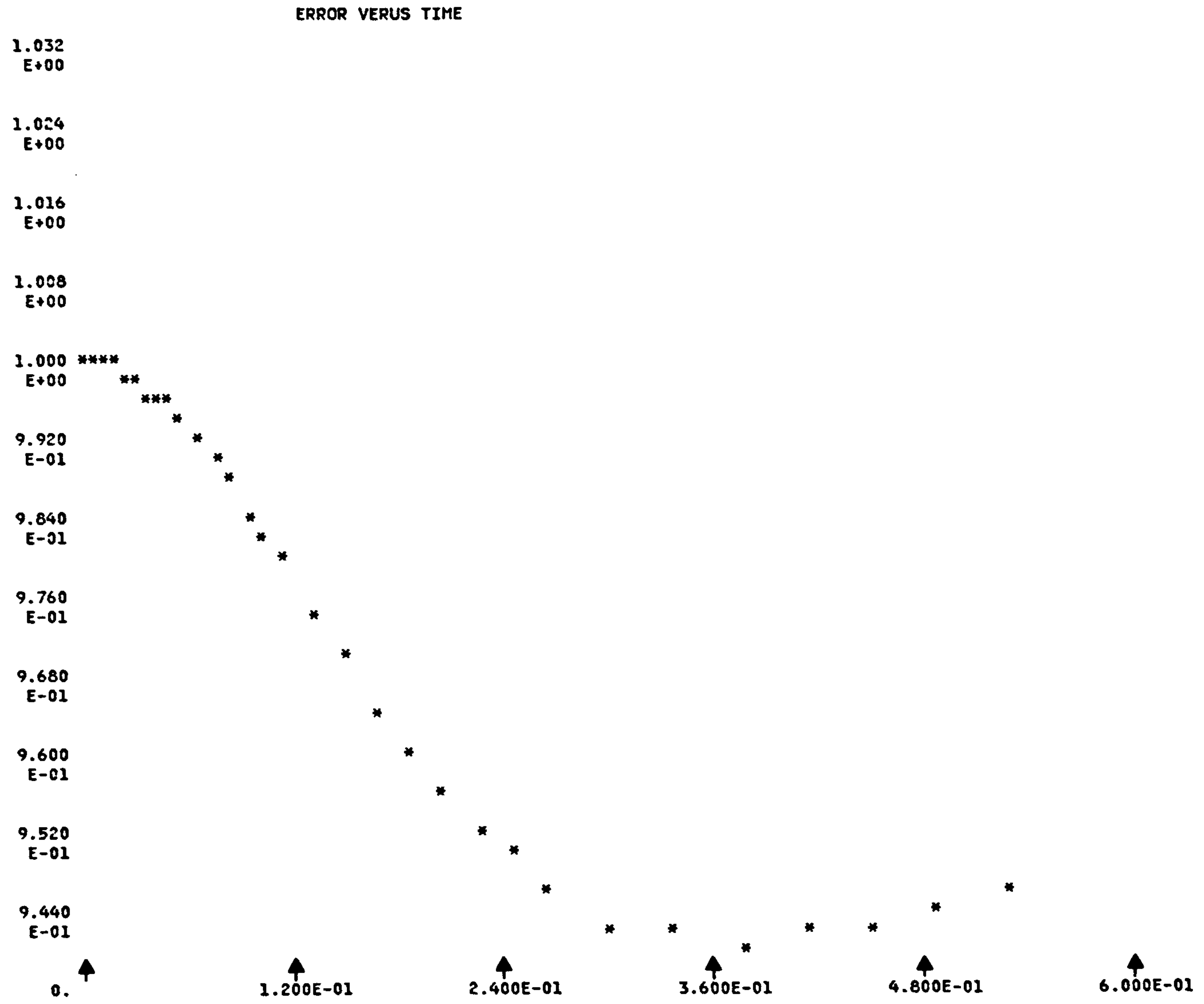
These variables are computed and printed automatically by ESP

TIME = actual execution time
MAXSTEP = maximum stepsize used
MINSTEP = minimum stepsize used
NUMSTP = number of integration steps taken
PLOT PTS = number of plot points stored (includes T = 0)
NTAP11 = number of plot points actually written onto TAPE11

OUTPUT VERSUS TIME



Typical graphic output using * GRAPH command with all default options



TIME = .056 SEC. time required for plotting

```
* AEROSPACE CORP 7000 SCOPE 2.1 LVL 270-E *       06/07/79      79158

SYS DEVICES 819/ 4/PF FLS=200K FLL=764K MXS=160K MXL=320/ MXB=320B

HH.MM.SS CPU SECOND ORIGIN
         10.59.58.MFA. **    NOS/BE 1.2 461A05*A                       05/01/79
13.54.07 00000.007 MFZ.     -A185T,STMFZ ,P2000, MT01, NT01
13.54.07 00000.008 JOB.     -ACCOUNT(    WERTZ, H J       04606  185T33170J  A12071ESPTST86457 532121
13.54.07 00000.014 JOB.     -ATTACH,LIB1,2NEWRESP.
13.54.07 00000.019 MFZ.        PF254 - CYCLE   16 ATTACHED FROM SN=SYSTEM
13.54.07 00000.020 JOB.     -ATTACH,LIB2,3FTNPLOTLIB.
13.54.07 00000.025 MFZ.        PF254 - CYCLE   15 ATTACHED FROM SN=SYSTEM
13.54.07 00000.026 JOB.     -LIBRARY,LIB1,LIB2.
13.54.08 00000.045 LOD.     -PRECOMP.
13.54.09 00000.220 MFZ.        LD610 - FLS REQUIRED TO LOAD - 0023330 OU.COG
13.54.09 00000.222 MFZ.        LD603 - EXECUTION INITIATED OS.EXP
13.54.09 00000.222 USR.       FORTRAN LIBRARY 452           02/09/78
13.54.09 00000.253 USR.           END  PRECOMP
13.54.09 00000.253 USR.            .030 CP SECONDS EXECUTION TIME
13.54.09 00000.254 LOD.     -FTN,I=IMSORC.
13.54.16 00000.766 USR.            .493 CP SECONDS COMPILATION TIME
13.54.16 00000.767 LOD.     -LGO.
13.54.35 00001.123 MFZ.        LD610 - FLS REQUIRED TO LOAD - 0023330 OU.COG
13.54.35 00001.125 MFZ.        LD603 - EXECUTION INITIATED OS.EXP
13.54.35 00001.126 USR.       FORTRAN LIBRARY 452           02/09/78
13.54.35 00001.127 USR.      .MAPS.1ESP
13.54.35 00001.218 USR.        EXIT
13.54.35 00001.219 USR.            .092 CP SECONDS EXECUTION TIME
13.54.35 00001.219 LOD.     -PRINT(OUTPUT,CHARS1=UC5A,CLASS=Y)
13.54.37 00001.332 JOB.     -COMMENT.......PRINT UTILITY
13.54.37 00001.339 JOB.     -ATTACH,YYYYYPR,3YYPRINT,ID=11673.
13.54.38 00001.344 MFZ.        PF254 - CYCLE    2 ATTACHED FROM SN=SYSTEM
13.54.38 00001.345 LOD.     -IFE,FILE(OUTPUT,.NOT.AS.OR.BOI),CONT.
13.54.38 00001.367 USR.      ELSE,CONT.
13.54.38 00001.368 LOD.     -REWIND,OUTPUT.
13.54.39 00001.333 LOD.     -COPY,OUTPUT,YYYYYTM.
13.54.39 00001.491 USR.        UT031 - EOI ENCOUNTERED
13.54.39 00001.492 USR.        UT035 -     EOR - 581          EOS - 1              EOP - 1
13.54.39 00001.493 LOD.     -REWIND,OUTPUT,YYYYYTM.
13.54.39 00001.508 LOD.     -YYYYYPR(OUTPUT, ABORT, A=Y, UC5C=UC5A/****/****/****,
13.54.39 00001.509 JOB.     -         ASC1=ASC1,   =, STD1=STD1, $****$=****, 0=0,
13.54.39 00001.509 JOB.     -         001=001, N=N, 060=*)
13.54.39 00001.513 USR.       FORTRAN LIBRARY 428           11/17/77
13.54.39 00001.516 USR.          STOP
13.54.39 00001.517 USR.            .004 CP SECONDS EXECUTION TIME
13.54.39 00001.517 LOD.     -COPY,YYYYYTM,OUTPUT.
13.54.40 00001.623 USR.        UT031 - EOI ENCOUNTERED
13.54.40 00001.624 USR.        UT035 -     EOR - 581          EOS - 1              EOP - 1
13.54.40 00001.625 LOD.     -ENDIF,CONT.
13.54.40 00001.643 JOB.     -DISPOSE,OUTPUT,*PR=C99,ST=IBM.
13.54.40 00001.644 LOD.     -RETURN,YYYYYTM,YYYYYPR.
13.54.41 00001.660 LOD.     -REVERT.
13.54.42 00001.686 MFZ.        RM770 - MAXIMUM ACTIVE FILES                9
13.54.42 00001.686 MFZ.        RM771 - OPEN/CLOSE CALLS                   50
13.54.42 00001.686 MFZ.        RM772 - DATA TRANSFER CALLS             3,461
13.54.42 00001.686 MFZ.        RM773 - CONTROL/POSITIONING CALLS          77
13.54.42 00001.687 MFZ.        RM774 - BM DATA TRANSFER CALLS          1,871
13.54.42 00001.687 MFZ.        RM775 - BM CONTROL/POSITIONING CALLS      309
13.54.42 00001.687 MFZ.        RM776 - QUEUE MANAGER CALLS               463
13.54.42 00001.687 MFZ.        RM777 - RECALL CALLS                      399
```

Control cards used to run example case on CDC 7600.

Special control cards used to generate this listing for publication: not normally used for an ESP job.

Normal day file data.

```
13.54.42 00001.686 MFZ.       SCM                 19.445  KWS
13.54.42 00001.688 MFZ.       I/O                  0.029  MW
13.54.42 00001.688 MFZ.       RMS                  0.013  MWS
13.54.42 00001.688 MFZ.       USER                 0.129  SEC
13.54.42 00001.689 MFZ.       JOB                  1.691  SEC
13.54.42 00001.689 MFZ.       SC050 - 000452 SC/LC SWAPS
13.54.42 00001.689 MFZ.             CU      = HC + 8 * CPU MINUTES +    2.5  * 1/O
13.54.42 00001.689 MFZ.             0.798   = .5 +         0.225    +        0.072

------ JES2 JOB STATISTICS ------

       651 CARDS READ

         0 SYSOUT PRINT RECORDS

         0 SYSOUT PUNCH RECORDS

     0.00 MINUTES ELAPSED TIME
```

Continuation of normal day file.

SECTION III

DEFINING DERIVATIVES

The most important segment of coding which the user must provide is that which defines his derivative equations. This segment will be translated by the precompiler into SUBRØUTINE DERIVS which is then called as needed by the ESP integration package for each evaluation of the derivatives. The user, therefore, must code his equations in a manner that can be recognized and properly interpreted by ESP. There are several ways to do this, and the user can choose the one or more ways most suitable to his problem from the following alternatives:

- The derivatives may be written as a set of first-order differential equations in terms of the ESP variables Y(i), DY(i), and T.
- The derivatives may be written in terms of the user's variables and their values then "moved" into the ESP array, DY. (This option is particularly desirable--frequently necessary--if WHELP expressions are used to compute the derivatives.)
- The derivatives may be written as *BLØCK statements, which permit direct and simple translation of engineering block diagrams directly into code that ESP can interpret.

In any case, all derivatives must be defined within a section of coding which begins with the card *DERIVS starting in column 1 and is terminated with the card *ENDDERIVS, also starting in column 1. This section of coding may be placed first, after any user supplied subroutines, or it may be placed after the *ICCØMP... *ENDIC or *ØUTPUT... *ENDØUT sections. (See Appendix A-3 User's Deck Structure.) In addition to the derivatives, this section will contain any *SWTCH or *SWMEM cards used (See Section IV Discontinuities.).

A. DEFINING THE DERIVATIVES AS FIRST-ORDER DIFFERENTIAL EQUATIONS

SUBRØUTINE DERIVS is written by ESP from the segment of the user's coding beginning with the *DERIVS card and ending with the *ENDDERIVS card. It will receive from the integration package the value of the independent variable T and the vector Y containing the values of each i^{th} integral and must return to it the vector DY containing the derivatives of each corresponding Y(i). Therefore, certain points must be kept in mind in coding the derivative equations:

- Each equation must be solved for some $\dot{Y}$ so that it can be written in the form DY(i) = some expression, one DY(i) per integrator required.
- The variable T is always the independent variable. (Its range of values is specified on the *RUN card explained in Section VII-A.)
- Y(i) should always be assumed to represent the result of integration at the current value of T.
- DY(i) may appear on the right-hand side of an equation if it has been defined above. [If derivative equations interlock, i.e., DY(1) = function of (DY(2)), DY(2) = function of (DY(1)), these equations must be solved, either analytically or numerically to remove the interdependency before trying to integrate.]
- Variables other than T, Y and DY which are used in the expressions should be PAR's (see Section II-B, STEP 2-2), or variables defined in some way within this program segment.
- The exact number of derivatives to be integrated must be indicated on the *RUN card, explained in Section VII-A.
- To skip a derivative at any time, simply set DY(i) = 0, but make certain that neq, specified on the *RUN card, corresponds to the largest subscript of DY used, even though it is desired to actually integrate fewer derivatives.

The general form of the derivative equation coding is

DY(i) = some function of (T, Y, DY, PAR,
constants, user variables)

EXAMPLES:

1. $\dot{\theta} = \theta + 5.0t$ ⇒ DY(1) = Y(1) + 5.0*T

2. $\dot{Y} + bY = G_1(T-2.0)$ ⇒ DY(1) = -B*Y(1) + G1*(T-2.0)

   or DY(1) = -PAR(1)*Y(1) + PAR(2)*(T-2.0)

3. $$\left.\begin{array}{l}\dot{\varphi} = \theta + 2.0\varphi \\ \dot{\theta} = 0.5\dot{\varphi} + \text{COS}(\theta)\end{array}\right\} \Rightarrow \left\{\begin{array}{l}\text{DY(1) = Y(2) + 2.0*Y(1)} \\ \text{DY(2) = 0.5*DY(1) + COS(Y(2))}\end{array}\right.$$

4. $$\left.\begin{array}{l}\ddot{Y} + 2\xi w\dot{Y} + w^2 Y = a + b \cos (w_1 t) \\ \text{where } a = 10.,\ b = 3.,\ w_1 = 0.05,\ \xi = 0.5,\ w = 2.\end{array}\right\}$$

   $$\Rightarrow \left\{\begin{array}{l}\text{DY(1) = Y(2)} \\ \text{DY(2) = 10.0 + 3.0*COS(0.05*T)-2. *0.5*2.0*Y(2)-4.0*Y(1)}\end{array}\right.$$

B. DEFINING THE DERIVATIVES IN USER VARIABLES

Alternatively, the user may code his derivative equations using his own variable names if he especially wishes to keep them more easily recognizable or if he plans to use vector-matrix expressions coded in WHELP variables. To do so, however, he must place within the derivative segment but before the derivative equations, statements to move each integrator output (Y(i)) that he plans to use into his own variable location. Similarly, after his equations, he must move the values to be integrated into the DY vector. This may not be done with an EQUIVALENCE statement, because Y and DY are in the calling sequence of SUBRØUTINE DERIVS, and this would be a violation of FØRTRAN rules. It may be done using simple replacement statements or SUBRØUTINE MØVE, whose calling sequence is

```
CALL MOVE (A, N, B)
```

where

A is the first storage location from which data is to be moved. It may be specified as A(i), A(i, j), A(i, j, k), or simply A implying A(1, 1). Data will be transferred by columns.

N is the number of consecutively stored values tobe moved.

B is the first storage location to which data is to be moved, specified in the same way as A.

EXAMPLES:

1.
$$\left.\begin{aligned} \ddot{\theta} &= \gamma t + \dot{\theta} \\ \dot{\gamma} &= b\theta(t-5.0) \end{aligned}\right\} \Rightarrow$$

```
*DERIVS
   THETA = Y(1)
   THETADT = Y(2)
   GAMMA = Y(3)

   THTDTDT = GAMMA*T+THETADT
   GAMMADT = PAR(1)*THETA*(T-5.0)

   DY(1) = THETADT
   DY(2) = THEDTDT
   DY(3) = GAMMADT
*ENDDERIVS
```

2.
$$\left.\begin{aligned} \dot{z} &= 2.0{*}z - 3.5{*}\varphi_1 \\ \dot{\varphi}_1 &= \varphi_1 T + \cos\varphi_2 \\ \dot{\varphi}_2 &= \varphi_2 T + \sin\varphi_3 \\ \dot{\varphi}_3 &= \varphi_3 T + \cos\varphi_1 \end{aligned}\right\} \Rightarrow$$

```
*DERIVS
   DIMENSIØN PHI(3), PHIDØT(3)
C MØVE Ys INTØ USER VARIABLES.
   Z = Y(1)
   CALL MØVE (Y(2), 3, PHI)
C CØMPUTE DERIVATIVES.
   ZDØT = 2.0*Z - 3.5*PHI(1)
   PHIDØT(1) = PHI(1)*T + CØS (PHI(2))
   PHIDØT(2) = PHI(2)*T + SIN (PHI(3))
   PHIDØT(3) = PHI(3)*T + CØS (PHI(1))
C MØVE DERIVATIVES INTØ DYS.
   DY(1) = ZDØT
   CALL MØVE (PHIDØT, 3, DY(2))
*ENDDERIVS
```

3.
$$\left.\begin{aligned} &A(3,3),\ B(3),\ C(3) \\ &\dot{C} = A\ B \end{aligned}\right\} \Rightarrow$$

```
*DERIVS
   COMMØN/USERBLK/A
*IDECLARE A(3,3) B(3) CDØT(3)    $
   CALL MØVE (Y(1), 3, B)
   CDØT = A*B    $
   CALL MØVE (CDØT, 3, DY(1))
*ENDDERIVS
```

[Note on example 3: Refer to Appendix H on WHELP]

C. DEFINING THE DERIVATIVES AS ENGINEERING BLOCKS (*BLØCK)

Since engineering systems are often described by block diagrams (as in the example case, Section II-B) a special command card *BLØCK can be used to define the integration represented by a block diagram so that the user need not translate its contents into the form $\dot{Y}$ = some expression.

A block diagram of this form is commonly used to represent a first order filter in an analog system:

$$e_{in} \rightarrow \boxed{\frac{1}{s+\beta}} \rightarrow y(i)$$

It means simply that the output of the block equals the contents of the block multiplied by the input, or in this case

$$y(i) = e_{in}\left(\frac{1}{s+\beta}\right)$$

where 1/s represents an integrator. Solving this to remove the s, we get

$$y(i)\,(s+\beta) = e_{in}$$

or

$$\dot{y}(i) + \beta * Y(i) = e_{in}$$

or

$$\dot{y}(i) = e_{in} - \beta * y(i)$$

This could now of course simply be coded as

$$DY(i) = e_{in} - \beta * y(i).$$

However, the whole process of manipulating the variables (and thus possibly introducing errors) can be avoided by using the special format:

$$\boxed{\text{*BLOCK} \quad 1 \quad \beta \quad y(i) \quad e_{in} \quad \$}$$

where

1 denotes a first order block

β is a constant (which may be any legal FØRTRAN expression) <u>written without embedded blanks</u>

Y(i) is the dependent variable name for the output of the block

e_{in} is a FØRTF.AN expression for the desired block input (It may have blanks within it, and is terminated by the dollar sign.)

\$ is the required terminator.

All items are separated by blanks.

Second-order blocks can also be specified directly by using *BLØCK 2. Thus, a block of this form

$$e_{in} \longrightarrow \boxed{\frac{\alpha_1 s + \alpha_0}{s^2 + \beta_1 s + \beta_0}} \longrightarrow Y(i)$$

which is equivalent to

$$\ddot{Y}_i = \alpha_1 \dot{e}_{in} + \alpha_0 e_{in} - \beta_1 \dot{Y}_i - \beta_0 Y_i$$

can be coded in this format:

$$\boxed{\text{*BLØCK 2 } \alpha_1 \ \alpha_0 \ \beta_1 \ \beta_0 \ \text{Y(i) Y(j) } e_{in} \ \$}$$

where

2 denotes a second-order block

$\alpha_1, \alpha_0, \beta_1, \beta_0$ are constants (which may be any legal FØRTRAN expressions) <u>written without embedded blanks</u>

Y(i) is the dependent variable name for the output of the block

Y(j) is another dependent variable name for an intermediate variable [It will <u>not</u> be the derivative of Y(i)]

e_{in} is a FØRTRAN expression for the desired block input (It may have blanks within it, and is terminated by the dollar sign.)

\$ is the required terminator.

All items are separated by blanks.

NOTE

Variables used in *BLØCK statements must follow the same rules as those used in DY(i) = ... statements since all *BLØCK statements are translated by the precompile program into equivalent equations of the form DY(i) = ...

EXAMPLES:

1. $$Y(3) \rightarrow \boxed{\frac{1}{s}} \rightarrow Y(4)$$

```
*BLØCK 1 0.0 Y(4) Y(3) $
```

2. $$Y(2)*PAR(7) \rightarrow \boxed{\frac{1}{s + \Omega Z}} \longrightarrow Y(3)$$

```
*BLØCK 1 ØMEGA*Z Y(3) Y(2)*PAR(7) $
```

3. $$3.0\Omega \rightarrow \boxed{\frac{s + 1}{s^2 + 5.0s + \Lambda}} \longrightarrow Y(2)$$

```
*BLØCK 2 1.0 1.0 5.0 LAMBDA Y(2) Y(3) 3.0*ØMEGA $
```

IV. INTEGRATION PACKAGE

SECTION IV

INTEGRATION PACKAGE

A. GENERAL INFORMATION

The basic integration package provided by ESP uses an algorithm known as Adams integration, which combines variable stepsize with variable order integration to provide a highly flexible and efficient problem solving capability. Three alternative integration packages are also available and may be selected at the user's discretion: they are fourth-order Runge-Kutta, second-order Runge-Kutta, and Hamming fifth-order Predictor-Corrector (which uses fourth-order Runge-Kutta as a starter). Each of these alternative integrators may be used with either a fixed or variable stepsize. (An explanation of the various algorithms used may be found in Appendix E.)

1. Options

Adams integration will be used unless overridden by the user. To select one of the alternative packages, simply place the following card before all others in your deck:

```
*METHOD        name
```

where

*METHOD begins in column 1
name   is RK4 for fourth-order Runge-Kutta
         or RK2 for second-order Runge-Kutta
         or PC for Predictor-Corrector

[See Appendix D-3-b if you write your own MAIN program.]

The Adams integration package seems to be highly successful with a great variety of problems and therefore should be tried first for most problems. Problems which require inputs at discrete intervals, however, may be more suitable for Runge-Kutta integration (see Section IV-C-4).

No matter which integration algorithm is used, control of integration is maintained by SUBROUTINE ESPCTL. It computes the initial stepsize (if variable), calls the specified integration routine, calls the switching routines, checks for occurrence of switches, and handles printing and storage of plot data.

2. Stepsize Selection

Immediately after the first call to SUBROUTINE DERIVS, ESPCTL will determine the initial H in one of the following ways:

- If both Y_i and $\dot{Y}_i$ are nonzero, then

$$H = \min_i \left(|Y_i/\dot{Y}_i| * 0.2 \right)$$

- If either Y_i or $\dot{Y}_i$ is zero for all i, then

$$H = \frac{TF - TO}{512}$$

- The user may define H in ICCOMP, and this H will be tried first (see below).
- The user may select fixed stepsize integration (with RK2, RK4, or PC only) by setting FIXSTP > 0 (see below).

After initial stepsize selection, stepsize control proceeds differently in each integration package and is documented in the description of each method given below.

3. User Control of Stepsize

In order to change any of the stepsize variables, the user should place the following common block in ICCOMP and use arithmetic statements to define those variables he wishes to change:

```
CØMMØN/STPCØN/HP, H, FIXSTP, HMIN, HMAX
```

where

HP — is the current printing interval. This is normally changed from the *RUN card but may be changed by the user's program during the run and must be > 0.

H — will be the initial stepsize tried, and the current stepsize during execution. It may be set only in ICCOMP or at switching times and must satisfy HMIN ≤ H ≤ HMAX.

FIXSTP — (default = 0) if set > 0, causes H = FIXSTP at all times.

HMIN — (default = 0) the lower limit on the stepsize. HMIN > 0 causes printing of a warning message and continuation of integration with acceptance of errors whenever H < 2.0 × HMIN.

HMAX — (default = 1.0E50) the maximum stepsize permitted.

4. Controlling Solution Accuracy

Solution accuracy may be controlled by the user through either or both of two variables, which may be input on run-time data cards. The exact use of these variables depends upon the integration algorithm selected and is explained in the discussion of each given below. The cards should be placed after all derivative, input, and output coding but before the *RUN card. The formats are:

*EPS ϵ

*Q q_1 $q_2 \cdots q_n$ $

where

Default value of $\epsilon = 1.E - 6$

Default value of $q_i = 1.E - 10$

*EPS and *Q start in column 1

q_i can be either = a constant alone (assumed to be control for next variable)

or Qi = constant

or ALL = constant

$ is a required terminator for *Q

EXAMPLES:

```
*EPS     1.E-2
*Q       Q2 = 1.E-20       1.E-8       Q5 = 1.E5       $
*Q       ALL = 1.E-6       $
```

B. ADAMS INTEGRATION

The Adams integration package features both variable stepsize and variable order integration. Since it is able to dynamically adjust both the stepsize and the order according to the success of each integration step, it is capable of solving a wide diversity of problems with both accuracy and speed.

On the first call to ADAMS, an appropriate stepsize H is computed using the H supplied by ESPCTL as a starting point, a flag IFAIL is set to 0, the order K is set to 1, and all necessary coefficients of formulas are initialized and computed. On subsequent calls, IFAIL is reset to 0 and H retains the value assigned in the previous call and is merely tested to ensure that it is within the precision limits of the computer. Coefficients needed for integration are then recomputed only if H has changed.

From this point on, the sequence of events is the same for each call to ADAMS. A solution is predicted and DERIVS is called to evaluate the derivatives at the predicted solution. The local error is estimated at order K, K-1, and K-2, and if necessary the order is lowered for the next step.

If the errors are within an acceptable range, the step is considered successful, the predicted solution is corrected and the derivatives re-evaluated at T + H. Differences are updated for the next step, and the best order and stepsize are determined for the next step before control is returned to ESPCTL.

If the errors are not acceptable, T is reset to T-H, the flag IFAIL is incremented by 1 and then tested. On the first and second failures, H is halved and the order retained before retrying integration; on the third failure, the order K is also reduced to 1. If more than three failures occur, an optimum H is also computed before retrying. Again the size of H is tested, and if too·small for machine precision, the error tolerance is doubled, and KFLAG is set to 0 so that a warning message will be printed:

"REQUESTED ACCURACY NOT ACHIEVED AT T = ______.
REMAINDER OF SOLUTION IS SUSPECT."

The error control method of ADAMS utilizes both stepsize and order variation to keep

$$\left[\sum_{i=1}^{neq} \left(\frac{E_i}{Q_i} \right)^2 \right]^{1/2} \leq \text{EPS}$$

where

E_i = estimate of the error in Y_i mode in the current step

Q_i = maximum thus far of Q_i and $|Y_i|$ (updated at the end of each integration step)

(See Ref. 6.)

C. RUNGE-KUTTA INTEGRATION

Both fourth-order Runge-Kutta and second-order Runge may be used with either a fixed stepsize or a variable stepsize. Each is implemented in its own subroutine, but these routines are parallel in logic and sequence, and differ only in the integration equations and constants used. The following paragraphs refer to fourth-order Runge-Kutta with differences applying to second-order Runge-Kutta noted in parentheses.

1. Fixed Stepsize

If the variable FIXSTP is set > 0 by the user (see Section IV-A-3), the stepsize H = FIXSTP at all times, and no error testing of any kind is done. Each call to the integration routine causes two single integration steps, namely from T to T+H and from T+H to T+2H.

2. Variable Stepsize

If FIXSTP ≤ 0 (default is 0.), integration begins with the stepsize H as determined in ESPCTL. A double step is taken, from T to T+2H, and then two single steps are taken, from T to T+H and from T+H to T+2H. The total error is computed and compared with the permitted error bounds. If not acceptable, stepsize doubling is prevented and H is compared with HMIN to determine if the stepsize can be halved. If not, integration continues using the current H and a warning message is printed:

"REQUESTED ACCURACY NOT ACHIEVED AT T = _____,
REMAINDER OF SOLUTION IS SUSPECT."

If H can be halved, it is and the above process is repeated.

If the errors are within acceptable limits, they are further tested. If less than 0.5% (1% for RK2) of the error bounds, the stepsize H is permitted to double for the next integration step.

3. Error Control

The error allowed in the computation of Y(i)s is controlled by requiring that

$$\mathrm{EPS}^2 \geq \sum_{I=1}^{neq} \left[\frac{\mathrm{ERR\O R}(i)}{\mathrm{Q}(\mathrm{I})}\right]^2$$

and Q(I) is initially set to MAX(Q(I), |Y0(I)|) and then continuously updated to

$$\mathrm{Q(I)} = \mathrm{MAX}(\mathrm{Q(I)}, |\mathrm{Y(I)}|, |\underset{(\mathrm{T=T-H})}{\mathrm{Y(I)}}|)$$

The default value of EPS is 1.E-6 and of Q(I) is 1.E-10, but the user may change them to suit his problem. (See Section IV-A-4.)

4. Inputting Values at Discrete Intervals

Since the user sometimes wishes to input noise or compute values at discrete and predictable intervals during integration and because the number of evaluations of the derivatives is different for each integration routine, a special flag, FIRSTP, which signals the beginning of each integration step (or pair of steps) has been added. To use this flag, include the following card in the derivative segment of coding

```
CØMMØN/RKCØNT/FIRSTP
```

and test FIRSTP to determine when to input values. FIRSTP = 0. normally, but is set to 1.0 by the integration routine at the beginning of each step in the fixed step mode or at the beginning of each pair of steps in the variable step mode.

Simulations with noise may be easily set up by using *METHOD RK4 or RK2, setting FIXSTP ≠ 0. (See Section IV-A-3) and inputting a noise value whenever FIRSTP = 1.0 (See the example below.).

Alternatively, the automatic stepsize control feature can be retained and a crude kind of switching capability achieved by using *METHOD RK4 or RK2 with a variable stepsize and simply testing the FIRSTP flag (as shown in the example below). This will permit successful introduction of discrete or discontinuous values at the beginning of each integration step, irrespective of the step size.

EXAMPLE:

```
*METHOD RK4
*DERIVS
               CØMMØN/RKCØNT/FIRSTP
                   .
                   .
                   .
               IF (FIRSTP  .NE   1) GO TO 5
               ANØISE = RANF(0)
               Y(1) = Y(1) + ANØISE
5              CONTINUE
               DY(1) = ...
*ENDDERIVS
*ICCØMP
               COMMON/STPCON/HP, H, FIXSTP, HMIN, HMAX
                   .
                   .
                   .
               FIXSTP = 0.02
*ENDIC
```

D. PREDICTOR-CORRECTOR INTEGRATION

Hamming's fifth-order Predictor-Corrector is the algorithm used, but since it is not self-starting, fourth-order Runge-Kutta is used to start the solution at T_0 and to restart the solution after discontinuities or difficulties are encountered.

Using the stepsize H as determined in ESPCTL, steps 1-3 are taken with fourth-order Runge-Kutta and the errors are checked at T_1. If the error on this step exceeds the error bounds, the stepsize is halved and the solution is restarted from T_0. If the error is acceptable, step 4 is taken with Runge-Kutta and step 5 with Predictor-Corrector.

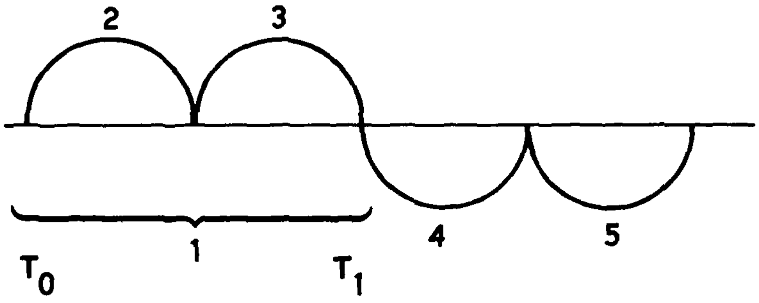


Fig. 1. Step Sequence for Starting Procedure

1. Variable Stepsize

Once the solution has been started in this way, error checks are made continuously, and the stepsize is halved when the error exceeds the bounds and permitted to double for the next step when the error is less than 1% of the bounds. To prevent excessive interval halving if the problem happens to be ill-conditioned, a counter is used to monitor stepsize halvings. Each time the solution is restarted with Runge-Kutta, the counter (KCØUNT) is started at zero. Each time the stepsize is decreased, KCØUNT is decreased by one, and after each successful step it is increased by one, but never permitted to exceed zero. If KCØUNT becomes less than -4, a warning message is printed: "SOLUTION APPEARS ILL-CONDITIONED AND IS BEING RESTARTED. THE

FOLLOWING Y'S EXCEED THE ERROR BOUND", followed by the Y(I)s. T is reset to T-H, and the solution is restarted using Runge-Kutta.

The above description of stepsize and error control assumes that the variable HMIN has its default value of zero. If, however, HMIN is set greater than zero (See Section IV-A-3) and the error is too large, the stepsize H is halved if possible and the solution continued. But, if it cannot be halved without making it less than HMIN, it retains its value, the solution continues and a warning message is printed: "REQUESTED ACCURACY NOT ACHIEVED AT T = ______, REMAINDER OF SOLUTION IS SUSPECT." This option, it may be seen, may produce less accuracy but in some cases more speed. The user is advised to consider any warning messages he receives and to base his selection of HMIN on the nature of his problem and the desired results.

2. Fixed Stepsize

Although the Predictor-Corrector integration package is intended for use as a variable stepsize method, it can also be used with a fixed stepsize by setting FIXSTP > 0. (See Section IV-A-3.). In this case, all error checking is skipped and no interval halving or doubling occurs.

3. Error Control

The error allowed in the computation of Y(i)s is controlled by requiring that

$$\text{EPS} > \frac{|\text{ERRØR(I)}|}{\text{Q(I)}} \qquad \text{for all I}$$

and Q(I) is initially set to MAX(Q(I), | Y0(I) |) and then continuously updated to

$$\text{Q(I)} = \text{MAX(Q(I)}, |\text{Y(I)}|)$$

The default value of EPS is 1.E-6 and of Q(I) is 1.E-10, but the user may change them to suit his problem (See Section IV-A-4).

SECTION V

DISCONTINUITIES

In programming a system of differential equations, the need frequently arises for a means to model accurately various types of discontinuities and nonlinearities which are part of the system. Therefore, several special features have been built into ESP to handle the most common types of nonlinearities, and with the aid of a little imagination (some examples will be given) nearly any desired characteristic can be produced by modifying one of the features.

It is important to understand that these features are provided not merely for convenience, however! Since the integration algorithms available with ESP by and large assume that they are working on continuous and reasonably well-behaved derivatives, haphazard introduction of discontinuities by the user can cause enormous problems and errors. The user is strongly urged to be certain that discontinuities are introduced only by means of one of the devices documented below and that the constraints mentioned with regard to their use be closely observed. (Section IV-C-4 discusses one other method of introducing discontinuities, which may be used with Runge-Kutta integration.)

These special features, which may be used only in the *DERIVS segment, consist of SWTCH's, which detect sign changes in an expression and restart integration, SWMEM's, which represent hysteresis nonlinearities and also restart integration, and an EVENT locator, which detects and reports the occurrence and timing of any user-specified event but which does not affect the integration process. Basic use of these features, which is fairly straightforward, will be documented first, and the later part of this section will be devoted to extended usage, a description of how the switches work, and some details and considerations regarding timing and accuracy.

A. DETECTING A SIGN CHANGE (*SWTCH)

The *SWTCH command detects a change of sign in its control or input expression, locates the time of this change within a specified degree of accuracy (see Section V-F), assigns itself an output according to the sign of the input expression, produces an automatic print point at the switch time (which the user may suppress), and restarts the integration from the switch time. It is useful, therefore, in producing an accurate discontinuous driving function to a derivative equation and in permitting the user to detect the exact time of a switch and, if he wishes, to perform some specific act at that time. Other possible uses will be illustrated in the examples.

The general form of the *SWTCH command is

$$\boxed{\text{*SWTCH}\quad i\quad 0_+\quad \$\quad 0_-\quad \$\quad \text{control expression}_i\quad \$}$$

where

*SWTCH starts in column 1

i $(1 \leq i \leq 50)$ is the number of this switch

0_+ is any legal FØRTRAN expression which will be the output if the control expression > 0.

0_- is any legal FØRTRAN expression, which will be the output if the control expression ≤ 0.

control expression$_i$ is any legal FØRTRAN expression involving only T, Y, PAR, system functions and constants. (For use of other variables, see Section V-D-1.)

$\$$ is a required terminator of the 0_+, 0_-, and control expressions

The ESP precompiler breaks up the *SWTCH card coding into the input (control expression), which it writes as part of SUBRØUTINE SWINPT in the form VALUES(i) = control expression$_i$, and the output computation which it writes as part of SUBRØUTINE DERIVS.

There are two output variables available to the user resulting from the *SWTCH statement. The first, SWCHi, is available only in the derivative segment of coding. To use it elsewhere, such as in ØUTPUT, the user must compute it himself (see Section V-D-2). The second, SWTCH(i), is available in DERIVS, ØUTPUT, SWINPT, SWMEMN and in any other routine in which the common block SWTCHS appears. The variables contain the following information:

$$\text{SWCHi} = \begin{cases} 0_+ \text{ if control expression}_i > 0. \\ 0_- \text{ if control expression}_i \leq 0. \end{cases}$$

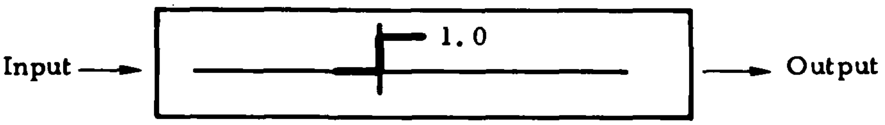
SWTCH(i) where: |SWTCH(i)| is one larger than the number of sign changes made thus far by the control expression, and is normally a floating point integer

The sign of SWTCH(i) is the current sign of the control expression

SWTCH(i) serves as a signal to the user that a switch has just occurred: On the first call to DERIVS following a switching, each SWTCH(i) which has been toggled has its absolute value increased by 1.5. See example 2 below for a way to utilize this trait, and see Section V-E for detail on the exact sequence of events when a switch occurs.

EXAMPLES:

1. The example problem in Section II shows a typical use of *SWTCH:



This is coded as

```
*SWTCH   1   1.0   $   0.0   $   Input   $
```

which produces the following results:

If input $\leq$ 0., SWCH1 = 0., SWTCH(1) = - N (N = number of sign changes + 1)

If input > 0., SWCH1 = 1.0, SWTCH(1) = + N

Also, when the switch is detected and located within HSW(1) of the time it occurs, then

SWCH1 = 0., if input $\leq$ 0.
= 1., if input > 0.

and SWTCH(1) = SIGN(|N| + 1.5, Input)

In this example, SWCH1 is the relevant output and is used as a term in the expression for DY(1)

DY(1) = G1*SWCH1 - B*Y(1)

2. To detect the exact time at which subroutines are to be called to redefine a number of program constants, the following arrangement could be used. Notice that no value is assigned to SWCH1 because the output variable of interest here is SWTCH(i), and that the coding makes use of the fact that SWTCH(i)'s are nonintegers exactly at switch times.

(assume PAR(1) = time$_1$, PAR(2) = time$_2$, etc.)

```
*SWTCH   1   0.   $   0.   $   T-PAR(1)   $
      IF(SWTCH(1)  .NE.  AINT(SWTCH(1)))CALL DUMDUM1
*SWTCH   2   0.   $   0.   $   T-PAR(2)   $
      IF(SWTCH(2)  .NE.  AINT(SWTCH(2)))CALL DUMDUM2
```

3. To produce a sample and hold at times $t_1, t_2, t_3, \ldots$ a similar but more abbreviated setup can be used, employing only one switch which will detect a sign change as each successive time is reached. Dimension a vector TSAMP of length N and store the desired sample times T_1, $T_2, \ldots T_N$ into it. Initialize SAMPLE, PAR(3) = t_1 and I = 1 and then use the following coding:

```
*SWTCH   4   0.   $   0.   $   T-PAR(3)   $
      IF(SWTCH(4)   .EQ.   AINT(SWTCH(4)))   GØ   TØ   5
C   (THIS SECTION WILL BE EXECUTED ONLY AT SWITCH TIMES.)
      SAMPLE = ...
      I = I + 1
      PAR(3) = TSAMP(I)
    5 CONTINUE
```

In this example it is assumed that $t_0 < t_1 < t_2 \ldots < t_n$. As execution begins, this switch is first toggled when t equals t_1 and at that time SWTCH(4) has 1.5 added to it so that the IF test will fail. Thus, SAMPLE is computed and PAR(3) is reset to the next sample time. Since immediately after this the .5 is stripped from SWTCH(4) this coding will be then bypassed until t reaches the new value of PAR(3). Note that this represents one of the few cases in which it is permissible to store a time-dependent value in PAR and to change the input to a switch in a discontinuous manner.

B. HYSTERESIS NONLINEARITIES (*SWMEM)

A hysteresis nonlinearity, of the general type illustrated in Fig. 2, can be modeled using the *SWMEM cards explained below.

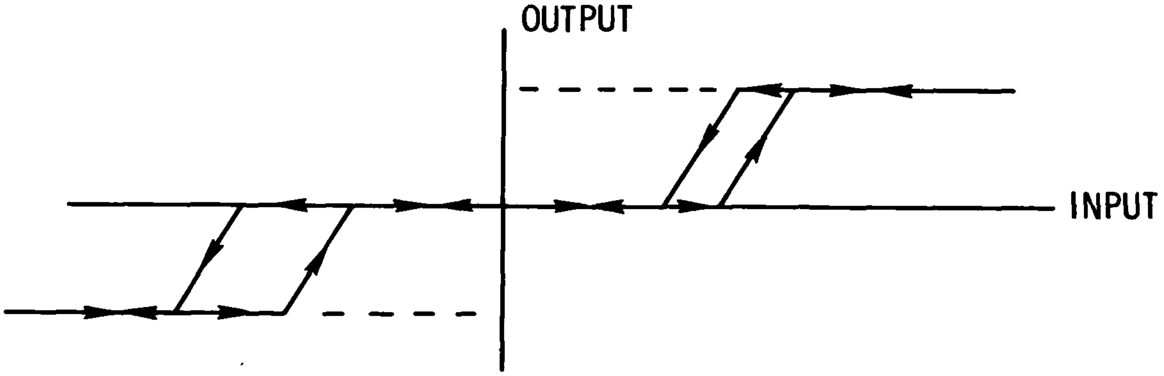


Fig. 2. General Form of SWMEM Nonlinearity

It will determine the proper location and output of the hysteresis within the required accuracy (see Section V-F), inform the user by setting a flag whenever a discontinuity occurs, indicate whether the output is in the linear, deadband or saturation regions, and restart the integration at each discontinuity.

There are three special control cards which may be used to implement this option:

*SWMEM (required) defines input to a SWMEM.

*SWMEMDATA (optional) defines the characteristics of the SWMEM.

*SWMEMSET (optional) initializes in saturation instead of at zero.

1. Defining Input to a SWMEM

The input to the hysteresis is defined on the *SWMEM card which is placed in the derivative segment of coding. The ESP precompiler will write the input as part of SUBRØUTINE SWMEMN in the form VALUES(i) = input$_i$. The format is

*SWMEM i input$_i$ $

where

*SWMEM starts in column 1

i (1 ≤ i ≤ 50) is the number of this SWMEM

input$_i$ is any legal FØRTRAN expression involving only the variables T, Y, PAR, system functions and constants. (See Section V-D-1 if other variables must be used.)

$ is a required terminator

EXAMPLES:

```
*SWMEM   1    Y(1) + Y(2) - CØS(PAR(12)*T)    $
*SWMEM   16    Y(3)**2 - PAR(1)*T/2.0    $
```

2. Defining Output of a SWMEM

There are two separate output variables from SWMEM's, parallel in nature to those from SWTCH's. The first, SWMi, is automatically computed and made available to the user in the derivative segment. To use it elsewhere, the user must compute it himself (refer to Section V-D-2). The second output variable, SWMEM(i, 4), is available in any routine where the common block SWTCHES appears. The variables contain the following information:

SWMi is the actual output value of the i^{th} SWMEM

SWMEM(i, 4) is normally a floating point integer indicating the present position on the hysteresis by its value:
-2.0 indicates negative saturation
-1.0 indicates slope on negative side[**]
0.0 indicates deadband
1.0 indicates slope on positive side[**]
2.0 indicates positive saturation

SWMEM(i, 4) also signals the user that a "corner has just been turned" on the hysteresis: On the first call to DERIVS following a SWMEM discontinuity, the absolute value of each SWMEM(i, 4) which has changed state is increased by 0.5. After the derivative equations are evaluated, the SWTCH's and SWMEM's are reevaluated and the 0.5 removed before the integration is restarted. (This signal may be tested and used in the same ways that SWTCH(i)'s are used in the examples (refer to Section V-E for more detail on the exact sequence of events.)

3. Defining the Characteristics of a SWMEM

Generally the user will want to define the constants C1 through C10 (see Fig. 3) characterizing his SWMEM, although they do have default values for the simplest case. Constants C3, C4, C8, and C9 are the slopes. However, an infinite slope is defined by setting the corresponding C_i equal to zero.

[**] If the user wishes to know what path he is following on the hysteresis, he may store the past value of SWMEM(i, 4) so he will know where he is coming <u>from</u> at each "corner."

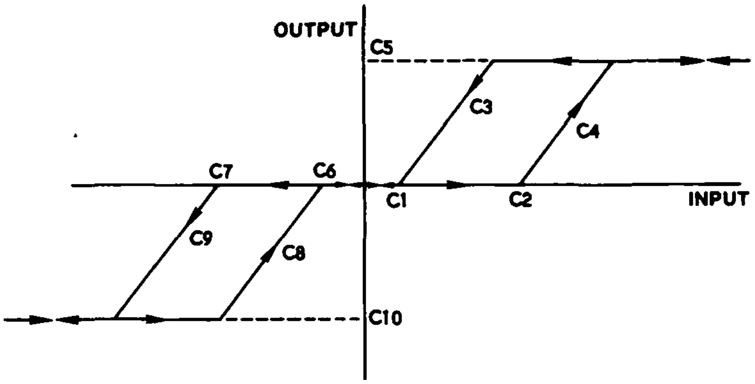


Fig. 3. SWMEM Characteristics

The C's are usually defined as part of the run-time data cards, meaning that they are placed somewhere between *ENDIC (if used) and *RUN, and are picked up in the same way that PAR's are picked up from *PAR cards. The format for inputting C's is

*SWMEMDATA

$$i_1 \quad c_1 \quad c_2 \quad \cdots \quad c_{10} \quad \$$$

$$i_2 \quad c_1 \quad c_2 \quad \cdots \quad c_{10} \quad \$$$

where

*SWMEMDATA starts in column 1

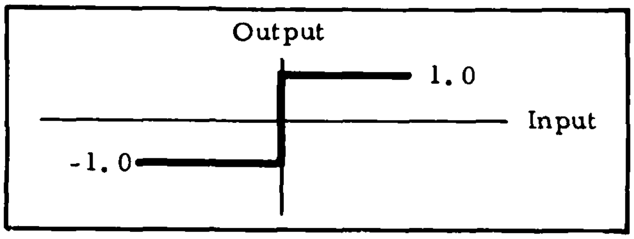
i is the number of the corresponding SWMEM

c_j is either a constant alone or C_j = constant, and blanks are separators. If a constant appears alone, it is assumed to be the value of the next c_j. If no value is given for c_j, its default value will be used.

$ is a required terminator

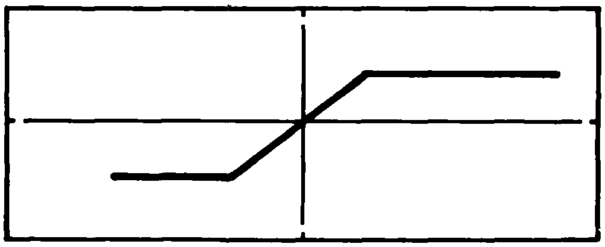
In general C1 must be > C6, except for special case 1 below.

Default values are C5 = 1, C10 = -1, all other Cs = 0., giving



SPECIAL CASES:

1. C1 = C2 = C6 = C7 is permitted, giving something like this:



2. For a symmetric nonlinearity (corresponding to forming quadrant III by rotating quadrant I through 180 degrees about the origin), only C1 through C5 need be defined with the result that

C6 = -C1 C9 = C4

C7 = -C2 C8 = C3

C10 = -C5

3. If it is necessary to compute any of the C values, they may be defined as CØNSTS in ICCØMP, if the following common block is added:

```
CØMMØN/SWHPAR/NCHNG, NALTER, ISWTYP, KSV,
CØNSTS(50, 10), SWSET(50)
```

C values defined in ICCØMP will supersede any that are input on *SWMEMDATA cards, but no error check will be made on them.

4. Initializing a SWMEM

Normally, the function is initialized on the region corresponding to starting from zero, but it may be initialized on the region corresponding to starting from saturation by using the *SWMEMSET card among the run-time cards. The format is

$$\boxed{\text{*SWMEMSET}\quad n_1\quad n_2\quad \ldots\quad n_\ell\quad \$}$$

where

*SWMEMSET starts in column 1

n_j is the number of the *SWMEM to be set and blanks are separators

$ is a required terminator of the list

The n_j are not retained from run to run, so it is necessary to redefine them for different runs.

C. LOCATING EVENTS WHICH DO NOT AFFECT INTEGRATION

The EVENT capability is useful for finding events which do not introduce discontinuities into the differential equations. It detects the time of occurrence of any event specified in the user-written SUBRØUTINE EVENTS within the timing accuracy specified on the *HSWE card (see Section V-F). The event is recognized as a change of sign in the input or event-defining expression, and any event can therefore occur many times within a run. Having located an event, ESP then interpolates T and Y to the event time and calls the user-supplied SUBRØUTINE NØTIFY, as its only response to the event. It does no printout and it does not in any way affect the integration process or the rest of the run.

To use the EVENT locator, three things must be done:

- Place an *NEVENT card among the run-time data cards:

```
*NEVENT    n
```

where

*NEVENT starts in column 1

n is the number of events to be defined

- Supply the subroutine to define the functions which determine the events:

```
SUBRØUTINE EVENTS(VALUES, T, Y)
DIMENSIØN VALUES (1), Y(1)

[Other common and dimension statements as needed]

VALUES(1) = expression determining event$_1$
 .
 .
 .
VALUES(n) = expression determining event$_n$
RETURN
END
```

- Supply the subroutine to receive notification that event number IEVENT has occurred at the given values of T and Y. This routine will be called once for each event occurrence, in the order in which events occur. Its format is:

```
SUBRØUTINE NØTIFY(T, Y, IEVENT)
DIMENSIØN Y(1)

[Other common and dimension statements as needed]

[Statements defining how EVENT information is to be used]

RETURN
END
```

EXAMPLE:

```
      SUBRØUTINE NØTIFY(T, Y, IEVENT)
      DIMENSIØN Y(1)
      PRINT 100, IEVENT, T
100   FØRMAT(1H0, *EVENT*, I4, *ØCCURRED AT*, E8.2)
      RETURN
      END
```

D. SWTCH's and SWMEM's: EXTENDED USAGE

Frequently the standard usage of SWTCH's and SWMEM's is too confining for the user's needs, either because of the limitations on how inputs may be defined or because the desired output values are not automatically available in all routines. The following paragraphs illustrate several ways that these limitations on both input and output can be circumvented.

1. Alternate Ways to Define SWTCH and SWMEM Inputs

Basic usage of *SWTCH and *SWMEM limits the form of their inputs to simple FØRTRAN expressions using T, Y, PAR, system functions and constants only, because of the way these statements are translated by the precompiler into the input routines SWINPT and SWMEMN. Since it is sometimes necessary either to use other variables or to execute a series of statements to define a switch input, alternate means of defining switch inputs are available. The first, and probably simplest, is for the user to write a function subprogram which defines his input; the second is to simply write the entire SWINPT or SWMEMN routine himself. In either case the user should remember that T and Y are always updated for the purpose of computing switch inputs, and that any time-dependent variables used to compute switch inputs should themselves be compute' within the function subprogram or within the user-written SWINPT (SWMEMN) so they too will be properly updated.

a. User-Written Functions

The major advantage in using function subprograms to define switch inputs is that it permits the user to combine the convenience of the *SWTCH(*SWMEM) card with almost total flexibility in defining his input. He may use common blocks to pass his own variables to the switch input computation, and he may use as many FØRTRAN statements as he wishes to define his actual input. The following example illustrates some possibilities of this approach:

EXAMPLE:

If the switch input is

```
if K1 = 1, input = Y(3) * 10. *BETA
if K1 = 0, input = Y(3) *BETA + Y(2)
```

and the desired output is

```
if input > 0, output = 1.0
if input ≤ 0, output = 0.0
```

the switch can be coded by putting the following cards in the derivative section

```
      CØMMØN/BLØCK1/K1,BETA
      .
      .
      .
      *SWTCH  1  1.0  $  0.0  $  SWFUNC(Y)  $
```

and writing the following function subprogram to be placed before the derivative coding

```
      FUNCTIØN SWFUNC(Y)
      CØMMØN/BLØCK1/K1,BETA
      DIMENSIØN Y(1)
      IF (K1 . EQ. 0) GØ TØ 5
      SWFUNC = Y(3) * 10. *BETA
      RETURN
 5    CØNTINUE
      SWFUNC = Y(3) *BETA + Y(2)
      RETURN
      END
```

b. User-Written SWINPT and SWMEMN

If the user has many switch inputs which require extensive computation, he may prefer to simply write his own input routines rather than write many functions. As with the function subprograms, any number of common blocks and computations may be included when the user writes his own subroutines SWINPT or SWMEMN. He must, however, write these routines in the form in which ESP expects them, as explained below, and be careful to place them after his job control cards but before the *DERIVS segment, which will result in their being used instead of the dummy routines written by the precompiler. Either routine or both may be user-written, but if the user writes his own routine for SWTCH inputs (SWMEM inputs), he must define all of his SWTCH inputs (SWMEM inputs) within it. Since his routine will supplant that written by PRECØMP, the effect of any input expressions coded only on *SWTCH (*SWMEM) cards will be lost. The procedure for writing SWINPT and SWMEMN is almost identical and is outlined in the following steps:

- Place the card, *SWITCHES n, (*SWMEMCNT n) where n is the number of SWTCH's (SWMEM's) being used, among the run-time data cards.
- Write the first four cards of SUBRØUTINE SWINPT (SUBRØUTINE SWMEMN), normally written by the precompiler, exactly as they appear in Appendix D-3-b.
- Include any common blocks, dimension statements, function definitions, and computations needed to define the switch input expressions.
- Define each of the SWTCH (SWMEM) inputs in the form

  VALUES(i) = input expression$_i$
- Conclude the subroutines by writing the cards RETURN and END.

2. User-Computed SWTCH and SWMEM Output

The switch output variables SWTCH(i) and SWMEM(i,,j) are passed in the common block SWTCHS to those segments of the coding translated by the precompiler. Therefore, they are available at any time within the subroutines DERIVS, ØUTPUT, SWINPT, and SWMEMN. SWCHi and SWMi, however, are computed only within the derivative segment and appear there only if *SWTCH and *SWMEM statements have been included. [If *SWTCH (*SWMEM) cards are used in addition to user-written SWINPT (SWMEMN), PRECØMP will correctly write the coding into DERIVS to define SWCHi (SWMi), even though the input expressions will be lost.] There are situations, therefore, in which the user may have to use SWTCH(i) and SWMEM(i, j) to compute SWCHi and SWMi himself: the first is during routine usage of *SWTCH or *SWMEM when the user wants to compute SWCHi or SWMi to use outside of DERIVS, and the second is when he has written his own switch input routines, uses no *SWTCH or *SWMEM cards, and wants to use these variables anywhere.

SWCHi can be computed in any routine which contains the common block SWTCHS by including the statements

```
IF (SWTCH(i) .GT. 0) SWCHi = some expression
IF (SWTCH(i) .LE. 0) SWCHi = some expression
```

SWMi can also be computed in any routine containing the common block SWTCHS by using the expression

```
SWMi = SWMEM(i, 3)-SWMEM(i, 2)*(SWMEM(i, 1)-input)
```

E. HOW THE SWITCHES WORK

If switches of any form, SWTCH, SWMEM, or EVENTS, are used in an ESP program, their inputs are processed regularly after the completion of successful integration steps to see if any switches have occurred. To minimize the calculations required to do this, the inputs are all defined in the separate routines--SWINPT, SWMEMN, and EVENTS--so that only these routines (and not DERIVS) need to be called to check the inputs. To detect switches and find the zero crossings, one past value of the input is always saved.

Once a switching has been detected, a modified version of Wilkinson's method (Ref. 2) is used to find the time of switching. If neither the saved value nor the present value of the input is zero, linear interpolation is used four successive times, testing for convergence each time. If convergence has not been accomplished after four iterations, a bisection is performed, and the linear interpolation is repeated in sequences of four plus a bisection until convergence is achieved. The condition for convergence is that the zero crossing be found within an interval of time (see the explanation of the *HSW, *HSWM, and *HSWE cards, Section V-F).

After determination of the first switching (if more than one occurred in the interval T - H to T), all switches which would switch within the accuracy requirements of the first are allowed to do so; then the solution is restarted. The sequence is as follows:

1. The zero crossing is found.
2. NØTIFY is called if EVENTS is used and any have occurred.
3. Printing is done at any print intervals prior to the switch time.
4. Variables to be used in restarting the integration are recomputed at the switch time.
5. If NDISPR = 2, the output data prior to the switching (that is, the data at the switch time but before the effect of the switch has been computed) is plotted and printed.
6. SWTCH(i)'s and SWMEM(i, 4)'s are set to the proper signs and values and 0.5 is added to those that have switched.
7. DERIVS is called to evaluate the derivatives with the 0.5 flags on the switch outputs (see Sections V-A and V-B-2).
8. All SWTCH and SWMEM inputs are reevaluated and their outputs updated in case one switch has toggled another.
9. EVENTS inputs are reevaluated if used.
10. The 0.5's are stripped from the switch outputs.
11. DERIVS is called to evaluate the derivatives without the 0.5's on the switch outputs.

12. Plot data is stored and if NDISPR $\neq$ 0, print occurs.

13. Integration is restarted from the switch time. (See flow charts of ESPCTL in Appendix D-4-c-iii.).

If another switch occurred later in the same integration interval, it will be detected after the next successful integration step, and the above procedure will be repeated.

This sequence of events has several important implications for programs in which two or more SWTCH's or SWMEM's occur in series:

- Two SWTCH's in a series (i.e., the first triggers the second) will produce the correct output for the second SWTCH in the second call to DERIVS. Also, SWTCH(i) will contain an accurate count of the actual number of switchings to date, but no flag will appear on the second SWTCH(i). Thus the "IF (SWTCH(i) .EQ. AINT(SWTCH(i)))..." test will not work for the second switch.

- Two SWMEM's in a series (i.e., the first triggers the second) will also produce the correct output for the second SWMEM in the second call to DERIVS. On both the first and second calls to DERIVS, the integer value of SWMEM(i, 4) will accurately indicate the position on the hysteresis. However, no flag will appear on the second SWMEM and, like the second SWTCH(i), the user will not be able to test for it.

- More than two SWTCH's or SWMEM's in a series will result in the first two being detected as above, and the next two being detected and reported some HSW or HSWM interval later, and so on.

- The amount and accuracy of print and plot data in the neighborhood of switches may be controlled by use of the flag NDISPR:

  NDISPR = 0 Plot data is stored at end of switch sequence, but no print occurs at the switch time unless it happens to coincide with a print time.

  1 Plot data is stored and print occurs at the end of the switch sequence.

  2 Plot data and print data are stored at the switch time both before the effect of the switch is calculated and afterward.

The default value of NDISPR is 1. To change this the user must include

```
CØMMØN/NDISPR/NDISPR
```

in his program (preferably in ICCØMP) and set NDISPR to the appropriate value.

F. CONTROLLING TIMING ACCURACY OF DISCONTINUITIES (*HSW, *HSWM, AND *HSWE)

There are three special control cards for controlling the allowable timing error in determining SWTCH's, SWMEM's, and EVENT's. All are optional and if used are placed among the run-time data cards, that is after *ICCOMP...*ENDIC and before *RUN, in any order. Their formats are

```
*HSW     h1   h2   ...  hn   $
*HSWM    h1   h2   ...  hn   $
*HSWE    h1   h2   ...  hn   $
```

where

Default values of all h_i = 1E - 6

*HSW (*HSWM, *HSWE) starts in column 1

h_i can be { a constant alone
Hi = constant
ALL = constant

(If a constant appears alone, it is assumed to be the control for the next SWTCH, SWMEM, or EVENT.)

$ is a required terminator

EXAMPLES:

```
*HSW            ALL=1.E-10   $
      produces:
                all HSW(i) = 1.E-10

*HSWM           H1=1.E-2    H3=1.E-10   $
      produces:
                HSWM(1)=1.E-2
                HSWM(2)=1.E-6   [default]
                HSWM(3)=1.E-10
                any additional HSWM(i)=1.E-6  [default]

*HSWE           H2=1.E-20    1.E-3    H6=1.E-10   $
      produces:
                HSWE(2)=1.E-20
                HSWE(3)=1.E-3
                HSWE(6)=1.E-10
                HSWE(1)=HSWE(4)=HSWE(5)=any additional
                  HSWE(i)=1.E-6   [default]
```

SECTION VI

OUTPUT

Output from ESP may take a variety of forms, depending upon the needs and wishes of the user. Printed output may be automatically formatted by means of the ESP command *PRINT or it may be tailored to the user's specifications using standard FØRTRAN. Graphic output may be produced on computer printout, on microfilm or on paper by the Calcomp pen plotter (see "IPD Computing Guide," Ref. 4). Data files may be written onto magnetic tapes for later use by another program or for later plotting. Any or all of these modes of output may be combined in any one program.

Because each mode of output has advantages for particular situations, the following sections will attempt to give the user sufficient information, not only to easily use each one, but also to help him decide which best suits his needs.

A. PRINTED OUTPUT: AUTOMATIC FORMATTING

The fastest and easiest way for the user to obtain printed output from his program is with the *PRINT command. With this option, the user needs only to name the variables to be printed and the labels to be assigned; he gives no print formats. Labels and values are printed in the order named, six columns to a page, in E-format. All labels are printed and then all values, so that if there are more than six of each, corresponding labels and values will not be adjacent, although their correspondence will still be clear (see example below). *PRINT may be used in two different ways, depending on the nature of the values to be printed.

1. Printing ESP Variables (*PRINT)

If all the values to be printed can be expressed in terms of T, Y, DY, PAR, constants, user supplied functions, or system functions such as SIN, COS, and SQRT (i.e., no other variable names are needed to define the values for output), then the fastest and easiest way to obtain printout is to use the ESP command card

$\text{*PRINT label}_1 = \text{expression}_1\ \$\ldots.\ \text{label}_n = \text{expression}_n\ \$\ \$$

where

*PRINT starts in column 1

label_i is 10 characters (8 on IBM) or less (with no embedded blanks) which will be used to label the value of expression at print times

expression_i is any FØRTRAN expression using T, Y, DY, PAR, constants, and system functions up to 1206 characters (No other variables may be used.)

$\$\ \$$ terminates the entire *PRINT statement (No continuation marks are used and all 72 columns may be used.)

NOTE

*PRINT may appear only once in a program *if it is used without* *ØUTPUT *and* *ENDØUT.

The *PRINT statement will be translated into FØRTRAN statements which will be written as part of SUBRØUTINE ØUTPUT in the form

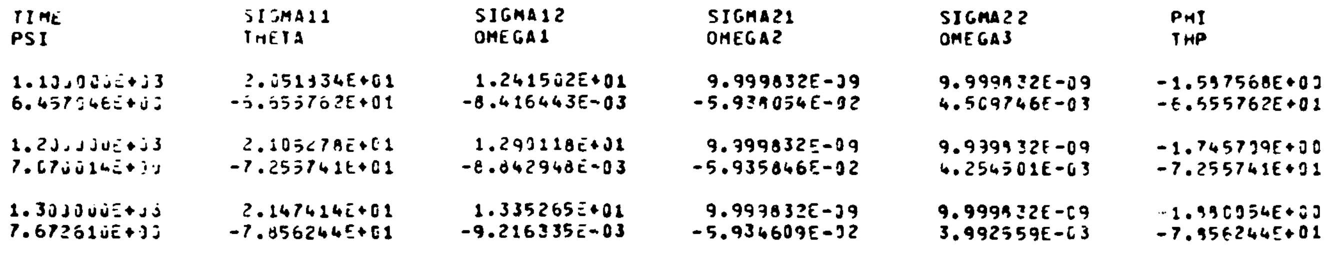
$$\text{PRINT(i)} = \text{expression}_i$$

The labels will be written onto a *HEADINGS card which will be placed on TAPE12 with the other run-time data cards. The number of labels found on the *HEADINGS card at execution time will determine the number of values actually printed from the PRINT vector (see *HEADINGS card, Section VII-G-1.)

EXAMPLE:

```
*PRINT    TIME=T  $  SIGMA11=Y(1)*PAR(1)  $
          SIGMA12=Y(2)*PAR(1)  $  SIGMA21=Y(3)*PAR(1)  $
          SIGMA22=Y(4)*PAR(1)  $  PHI=Y(5)*PAR(1)  $
          PSI=Y(7)*PAR(1)  $  THETA=Y(6)*PAR(1)  $
          OMEGA1=Y(8)*PAR(1)  $  OMEGA2=Y(9)*PAR(1)  $
          OMEGA3=Y(10)*PAR(1)  $  THP=(Y(6)-Y(11))*PAR(1)  $  $
```

produces a printout that looks like this:



```
TIME                SIGMA11             SIGMA12             SIGMA21             SIGMA22             PHI
PSI                 THETA               OMEGA1              OMEGA2              OMEGA3              THP

1.100000E+03         2.051334E+01        1.241502E+01        9.999832E-09        9.999832E-09       -1.597568E+00
6.457946E+03        -6.555762E+01       -8.416443E-03       -5.938054E-02        4.509746E-03       -6.555762E+01

1.200000E+03         2.105278E+01        1.290118E+01        9.999832E-09        9.999832E-09       -1.745739E+00
7.070014E+03        -7.255741E+01       -8.842948E-03       -5.935846E-02        4.254501E-03       -7.255741E+01

1.300000E+03         2.147414E+01        1.335265E+01        9.999832E-09        9.999832E-09       -1.990054E+00
7.672610E+03        -7.856244E+01       -9.216335E-03       -5.934609E-02        3.992559E-03       -7.856244E+01
```

2. Printing User Variables or Computing Output (*ØUTPUT...*ENDØUT)

To print the values of expressions which contain variable names other than T, Y, DY, PAR, system functions or function subprograms, the ESP command cards, *ØUTPUT and *ENDØUT must be used to signal the beginning and the ending of the output computations and printing commands. All statements between these two cards will be written into the output subroutine, and the user will find it helpful to remember that this program segment is a separate subroutine as he decides what he may and may not do within it. In general, between *ØUTPUT and *ENDØUT, any FØRTRAN or WHELP may be used to define user output variables, and *PRINT, as defined above, may be used to print them; but certain rules must be followed:

- If *PRINT and *ØUTPUT...*ENDØUT are both used, *PRINT may only be used between *ØUTPUT and *ENDØUT, but it may appear as many times as necessary here as long as the total number of print variables specified does not exceed 60.
- User variables appearing in *PRINT expressions must be defined before *PRINT.
- FØRTRAN rules regarding the sequence of declarative and executable statements within the section must be followed.

The general format of this section is

*ØUTPUT

FØRTRAN and WHELP statements (in any order consistent with FØRTRAN rules)

*PRINT label$_1$ = expression$_1$ \$... label$_n$ = expression$_n$ \$ \$

FØRTRAN and WHELP statements

*ENDØUT

EXAMPLE:

```
*ØUTPUT

    CØMMØN/BLØCKA/A, B, R, ØMEGA

*IDECLARE A(6, 3) B(3, 6) R(6) X(6) $        [WHELP statement]

    CALL DIDDLE (T, Y, THETA, ALPHA)

    X = A * B * R/ (THETA) $                 [WHELP statement]

    Z = THETA + T * ALPHA

*PRINT TIME = T  $  X1=X(1)  $  X2=X(2)  $

    X3=X(3)  $  THETA=THETA  $  Z=Z  $  $
    (additional FØRTRAN)

*ENDØUT
```

Notice that the subroutine DIDDLE must be supplied by the user, and that the variables A, B, R, and ØMEGA must be defined elsewhere.

3. Accuracy of Printed Values

Since the print interval and stepsize are unrelated, it is generally necessary to evaluate the solution just for printout. Adams integration uses the code and method outlined in Ref. 6 , and the solution (Y's) and derivatives (DY's) are interpolated from the difference table kept internally. For predictor-corrector and Runge-Kutta integration, in order to avoid calling the integration routine just for printout, Hermite interpolation (Ref. 3), which uses the functional values and their derivatives at three points, is employed to evaluate the solution (Y's) and derivatives (DY's) at intermediate points. Thus, the proper T(time) and its corresponding Y's and DY's are automatically passed to ∅UTPUT at a print time, and these values and any other output variables computed using only these and constants will be correct for the time given.

However, since in the process of integration ESP generally oversteps the print time and "backs up" to print, PAR's and time dependent variables which may have been passed to ∅UTPUT by a user-supplied common block may not correspond to the print time. The best way to avoid this problem is to recompute these variables within *∅UTPUT... *END∅UT, so that they will always reflect the actual values at the print time. If this involves much computation, the user may want to test PRINT(1), as shown in the example in Section VI-C, to ensure that these values are recomputed only for printout. (Since plot data storage occurs at the end of each successful integration step, or pair of steps, irrespective of the print interval, data stored for plotting will be consistent and the user need not concern himself with this problem.)

B. GRAPHIC OUTPUT

To produce graphic (plotted) output from an ESP run, the user has two tasks: storing the data to be plotted and specifying how it is to be plotted. The simplest way to do these tasks is to store data into the vector PL∅T and then use the *GRAPH command (explained below) to plot it. An alternative way is to write all plot data onto a magnetic tape or disk file and then plot it using some other plotting routine (see "IPD Computing Guide," Ref. 4). The first method is the simplest and most satisfactory for most user needs. The latter is useful mainly for very time-consuming program runs where it is desirable to have output data available for repeated plotting or study without the necessity of rerunning the program. (Refer to Section VI-D on Data File Output.) It is also possible to produce overlays using data generated in different runs (see Appendix F on Multiple Runs.).

1. Storing Plot Data

All data to be plotted by *GRAPH must be stored by the user in the vector PLØT. The simplest way to do this is within the *PRINT statement used with or without *ØUTPUT... *ENDØUT, as follows:

```
*PRINT TIME=PLØT(1)=T   $   RATE=PLØT(2)=Y(1)   $
    ØMEGA=PLØT(3)=Y(1) + Y(2)   $   $
```

This statement accomplishes both printing and labeling as explained above and storing of data for plotting. It generates the following statements in the FØRTRAN version of SUBRØUTINE ØUTPUT.

```
PRINT(1)=PLØT(1)=T
PRINT(2)=PLØT(2)=Y(1)
PRINT(3)=PLØT(3)=Y(1) + Y(2)
```

The user may also store the plot data himself if he is not using *PRINT or wishes not to bother adding to or changing his *PRINT cards. To do so he simply adds the FØRTRAN statements needed to the *ØUTPUT.. *ENDØUT section of his program. The example above would then become

```
*ØUTPUT
*PRINT TIME=T   $   RATE=Y(1)   $   ØMEGA=Y(1) + Y(2)   $   $
      PLØT(1)=T
      PLØT(2)=Y(1)
      PLØT(3)=Y(1) + Y(2)
*ENDØUT
```

WARNING

Do not attempt to store plot data in this way in other parts of the program (anywhere outside of *ØUTPUT.. *ENDØUT); array PLØT will not be recognized and the timing of data storage will be wrong.

The assumed or default number of plot variables which may be stored in this way is 10. If more than 10 variables are to be stored, a *MAXPLØTS card must be placed among the run-time data cards, that is, after *ICCØMP... *ENDIC, but before *RUN. The format is

```
*MAXPLØTS n
```

where

*MAXPLØTS starts in column 1

n is an integer (≤ 100) specifying the maximum number of plot variables to be used

During execution of the program, ESP uses a file (TAPE11) to store the data placed in the PLØT vector, storing a point at the end of each successful integration step or pair of steps and at each switch time. At plotting time ESP selects a representative sample of this data for actual plotting (up to 1000 values per variable), and in most cases no significant loss of information occurs. However, if the user wishes to have greater control over the number of points or the intervals over which they are plotted, PRØGRAM ESPPLØT may be used instead. (See Section VI-D-1.)

2. Plotting Output (*GRAPH)

Printer, film, and/or pen plots are obtained by placing *GRAPH commands after the *RUN which computes the results to be plotted.[†] The general form is

*GRAPH n_x n_y [size][grid][scaling][type]
[title]
[X title]
[Y title]
*GRAPH n_x n_y ......

where

n_x ($1 \leq n_x \leq 100$) is the PLØT subscript of the desired x variable

n_y ($1 \leq n_y \leq 100$) is the PLØT subscript of the desired y variable

([size], [grid], [scaling], and [type] are optional, and may appear in any order after n_x and n_y but must appear on the same card with *GRAPH.)

[†] Plotting is accomplished by the subroutine GRAPH; for more complex needs and some further options, the user is referred to the writeup of this subroutine.

More specifically

```
[size] = ┌SMALL    ┐
         │SIZExxyy │
         │OVERLAY  │
         └OVERLAY1 ┘
```

where

Default choice is SMALL

SMALL implies a 6 X 10-in. printer plot or a 10 X 15 in. hardcopy or film plot

SIZExxyy implies an xxXyy-in. plot (10 X 10-in. or 10 X 15-in. nicely fills a linear microfilm grid; this will be turned sideways and possibly cover more than one page on the printer plots.)

OVERLAY implies size and scaling as on the previous graph. A new plot will result on the printer while a true overlay will be made on pen or film plots (See TYPES).

OVERLAY1 implies the same as OVERLAY except that the data is completely rescaled and on pen and film a new y-axis scale is placed at the right end of the graph.

```
[grid] = [GRIDggg]
```

where

Default choice is GRID3A1

ggg is the three-character number specifying the type of plot grid to be used (see "IPD Computing Guide," Ref. 4)

Printer plots are always linear-linear

If grid is semilog or log-log, film/pen plots are made using log$_{10}$ of the X or Y data or both

$$\boxed{[\text{scaling}] = \left[\text{SCALE} \quad \left\{\begin{matrix}\text{AUTO}\\ X_o\ \text{XDEL}\end{matrix}\right\} \quad \left\{\begin{matrix}\text{AUTO}\\ Y_o\ \text{YDEL}\end{matrix}\right\}\right]}$$

where

Default choice is AUTO for both axes (automatic scaling based on actual data stored)

If SCALE appears, the parameters within $\{\ \}$ must be specified for both X and Y

AUTO produces automatic optimized scaling based on actual data for X-axis (Y-axis)

X_o (Y_o) is the minimum scale value to be used for X data (Y data)

XDEL (YDEL) is the absolute value of the difference between one scale annotation and the next. For a 10 × 10-in. plot, XDEL = (XMAX - X0)/10.

$$\boxed{[\text{type}] = [\text{TYPE}t_1[t_2][t_3]]}$$

where

Default choice is TYPEP

$$t_i = \left[\begin{array}{ll} \text{P} & \text{for printer plots} \\ & \quad\text{or} \\ \text{S} & \text{for printer plots to be overlaid} \\ \text{F} & \text{for film plots} \\ \text{C} & \text{for Calcomp pen plots}^{\dagger} \end{array}\right]$$

Any combination of P, F, and C may be used (Example: TYPEFC, TYPEP, TYPECFP)

S may be used in place of P if Printer plot overlays are to be made. If TYPES is specified on the *GRAPH card, no printer plot will be produced for that card, but the plot data will be stored. Thus, utilizing TYPES on all but the final overlay card will produce a single printer plot containing all overlays to be produced.

[†] See "IPD Computing Guide," Ref. 4, for other steps required to obtain pen plots.

EXAMPLE:

The following sequence of *GRAPH cards will produce a single printer plot with three graphs on it.

```
*GRAPH  1  2  TYPES
*GRAPH  1  3  TYPES  OVERLAY
*GRAPH  1  4  OVERLAY
```

$$[\text{title(s)}] = \begin{bmatrix} \text{plot title} \\ \text{X-title} \\ \text{Y-title} \end{bmatrix}$$

where

Default choice is blanks

Each title is any character string appearing in columns 1 through 50. (Column 1 should be blank to avoid possible misidentification.)

All titles are optional but must appear in the order given, each on a separate card (To delete one, substitute a blank card.)

EXAMPLES:

1.
```
*RUN    5   0   0.1   50
*GRAPH  1   4
```

This will produce a printer plot of the Y-axis data stored in PLØT(4) versus the X-axis data stored in PLØT(1). It will make a SMALL (6 x 10-in.) plot with automatic scaling and no titles. This represents the minimum specifications to produce a plot.

2.
```
*RUN    7   0   0.5   30
*GRAPH  2   5   SIZE1510   TYPEPF  SCALE  AUTO  5.0  50
                ERROR IN ANGLE OF ATTACK
                DESIRED ANGLE
                DIFFERENCE BETWEEN GOAL AND ACTUAL
```

This example shows considerable user control of plot parameters. It will produce a linear plot on printer <u>and</u> film with an X-axis 15-in. long and automatically scaled. The Y-axis will be 10 in. with Y_o at 5.0 and Y_{max} at 505.0. "ERROR IN ANGLE OF ATTACK" will be printed across the top of the plot, "DESIRED ANGLE" along the X-axis and "DIFFERENCE... " along the Y-axis.

C. PRINTED OUTPUT: USER FORMATTED

At any time the user wishes to produce output printed to his own specifications, he may do so by simply adding FØRTRAN print (or write[¶]) statements and their corresponding formats to any section of his program, even if he is also using *PRINT or *ØUTPUT...*ENDØUT. Since considerable care must be exercised in controlling the frequency of printout produced in this manner, it is suggested that *PRINT be used as much as possible during program development and debugging stages because of its speed and ease, and that FØRTRAN print statements be added only when a more specific format is needed for production runs. In adding print statements the user should consider the following points:

1. Since each segment of the user's coding results in a separate subroutine, the user must be careful to print in each segment only those values known within the section; that is, those values passed to it by ESP, defined within the segment, or passed to the segment by user-supplied common block.

2. A print statement placed within *ICCØMP..*ENDIC will be executed once at the beginning of each run, since that is the only time SUBRØUTINE ICCØMP is called. This is the logical place, therefore, to print variables which are constant for the particular run. (Refer to Section II-B-STEP5 for a list of the inputs and constants ESP prints automatically at program initiation.)

[¶] Since the main program written by PRECØMP does not assign a TAPEn type name to the Output file, if WRITE statements are used, the user must write his own main program in order to set TAPEn = OUTPUT.

3. Placed within *DERIVS... *ENDDERIVS, print statements will be executed every time the derivatives are evaluated, which is several times per step at possibly decreasing times, the exact number depending on the integration algorithm used. Printout of this type may be useful for specific debugging information, but will be unwieldy and confusing for general output.

4. Data which is to be printed only at the completion of a run may be printed from the MAIN program. The user would need to (1) place a *RETURN card at the end of the run-time data cards, to cause program control to return to MAIN; (2) write his own MAIN (see APPENDIX D-3-b) adding common blocks to transmit the data to be printed; and (3) add print coding to MAIN between CALL ESPII and END.

5. Within *ØUTPUT..*ENDØUT is generally the best place to add print statements, since the output subroutine is called at regular and predictable intervals, namely, once at the end of each integration step for plotting purposes and once at each print time for printing purposes. As long as printing is accomplished by *PRINT commands, ESP checks to see that printing occurs only at the specified print intervals. When the user writes his own print statements and formats, however, he must be the one to see that they are executed at the proper times. To do this he simply tests PRINT(1) <u>(before</u> setting it!). PRINT(1) will be 4HPLØT if ØUTPUT is being called for plotting purposes or 5HPRINT if it is called at a print time.

<u>EXAMPLE:</u>

```
*OUTPUT
     If (PRINT(1) .NE. 5HPRINT) GO TO 5
     PRINT 100, T, Y(1), DY(1)
100  FORMAT (1H0, 3E20.8)
  5  CONTINUE
       .
       .
       .
*PRINT ...
       .
       .
       .
*ENDØUT
```

D. DATA FILE OUTPUT

In addition to printed and plotted output from an ESP program, the user may wish to output his data on a magnetic tape that he can reuse after the termination of his job. Two common reasons for needing this capability are the desire to use the output as input to another computer program and the wish to ensure that data from a particularly time-consuming run is saved for plotting. In either case, there are two alternative ways to save these results, either by saving TAPE11, the file on which ESP automatically writes all plot output, or by saving a file created and named entirely by the user. TAPE11 is easier to produce, but because the data on it is both packed and blocked, a special program, ESPPLØT, must be used to plot it (see below). On the other hand, a user-created file can be tailored specifically for its later use, but requires more effort to generate.

1. Data Written onto TAPE11

ESP automatically writes all data stored in PLØT (either via *PRINT statements or by PLØT(i) = expression, refer to Section VI-B-1) onto a logical file named TAPE11, which is saved until run completion, plotted from by *GRAPH, and then released. To save this data after job termination, the user must transfer it to a magnetic tape. Once stored on magnetic tape, the data may be plotted at any time using program ESPPLØT.

ESPPLØT is a compiled (binary) main program which is to be run at some time later than the ESP run whose data it is to plot. It may be used to plot up to 10,000 points and allows the user to determine the time intervals to be plotted. Every point or every nth point may be plotted, and symbols may be placed on the plots at any Δt interval specified by the user. Steps required for its use are the following:

a. Make certain that the first element of the array PLØT stored by the ESP run is time (T).

b. Request the actual tape containing the data and assign it the logical name TAPE11.

c. Execute program ESPPLOT (See control cards in example below.)

d. Write the necessary *Control cards according to the formats given below.

ESPPLOT *CONTROL CARD FORMATS: All *Control cards must start in column 1. Non-* cards may occupy columns 2 through 72, except title cards, which are limited to columns 1 through 50.

*MAXPLOTS n

(n must be the same as n on the *MAXPLOTS card appearing in the ESP run. Default value is 10.)

*NLOCAL n

(Every nth data point is to be plotted. Default value is 1. May be changed at any time.)

*IMAX m T_{min_1} T_{max_1} ... T_{min_m} T_{max_m}

or

*IMAX m ALL T_{min} T_{max}

(The next m sets of *GRAPHS are to be plotted for the m sets of intervals given. T_{min_n} and T_{max_n} apply to the nth *GRAPH encountered. If the m sets of graphs are all to be plotted for the same T_{min} and T_{max}, enter ALL followed by T_{min} and T_{max}. To change this, use *RETURN first: see example.)

*TICKPLOTS Δt

(The symbol ∇ will be placed on the plotted line every Δt seconds. The default is no symbol.)

*GRAPH nx ny [type] [size] [grid] [scaling]
[Title]
[x-title]
[y-title]

(See *GRAPH description, Section VI-B-2 above.)

*RETURN

(End this segment of instructions and begin another.
*IMAX, *TICKPLOTS, and *GRAPH must be reentered.
*NLOCAL will retain its value but may be changed.
*MAXPLOTS will retain its value.)

EXAMPLE:

```
ATTACH, ESPPLOT, 2ESPPLOT.
ATTACH, SUBLIB, 2ESPFTN.
ATTACH, PLOTLB, 3FTNPLOTLIB.
LIBRARY, SUBLIB, PLOTLB.
(Request  TAPE11  saved by ESP run)
ESPPLOT.
HARDCPY.
EOR
*MAXPLOTS  12
*NLOCAL  10
*TICKPLOTS  0.2
*IMAX  2  ALL  0.  10.
*GRAPH  1  9  TYPEF
    ETA  VS  TIME
*GRAPH  9  10
    ETADOT  VS  ETA
*RETURN
*NLOCAL  20
*IMAX  3  50.  100.  50.  100.  0.  100.
*GRAPH  1  2  TYPEF
*TICKPLOTS  5.
*GRAPH  1  3  TYPEF  OVERLAY
    THETA AND TAU VS TIME
*NLOCAL  10
*GRAPH  2  5  TYPEF
    X VS THETA
      THETA
        X
*RETURN
EOF
```

In the above example ESPPLOT expects TAPE11 to have 12 variables stored on it, of which the first is time. Selecting every 10th data point and placing a tick every 0.2 seconds, it will produce filmplots of variables 9 vs time and 10 vs 9 from T=0. to T=10.seconds and label them ETA VS TIME and ETADOT VS ETA, respectively.

A *RETURN card appears next so that IMAX can be changed. Note that IMAX in this section specifies 3 time intervals and therefore 3 *GRAPH cards follow it. However, only 2 plots will result, as the first two are overlaid. Selecting every 20th plot point, it will plot variable 2 vs time from time = 50. to 100. with no tick marks. Then it will overlay variable 3 vs time on the same grid with tick marks every 5. seconds and label this plot THETA AND TAU VS TIME. Finally, using every 10th data point, it will plot variables 5 vs 2 from time = 0. to 100. with tickmarks every 5. seconds, and this plot will have X VS THETA on top, THETA on the X-axis and X on the y-axis. A final *RETURN card indicates the end of the job.

2. Data Written onto User-Named File

To produce a saved data file which matches some particular user format, the user must declare and write his own file. To do this the following steps are necessary:

- Request that a magnetic tape be allocated to the ESP program and given a logical file name by using the proper control cards.
- Declare the logical file name by supplying the main program (ESP's PRECØMP normally writes it), and adding the logical file name to the PRØGRAM card (see example below).
- Write desired data on the file by using an unformatted WRITE(lfn) list statement within *OUTPUT... *ENDOUT. Remember that the output subroutine is called at various times for plotting and printing (refer to Section VI-C), and that any time the user does his own WRITE statements he must consider the frequency with which he wants to "write" and allow for it by testing PRINT(1) and acting accordingly.

EXAMPLE:

```
     [control cards]
     [control card additions to request and save a tape and to name it
     TAPE20, according to current operating system manual.]
7-8-9
     PRØGRAM MAIN(TAPE12,TAPE11,ØUTPUT,INPUT=TAPE12, TAPE20)
     EXTERNAL DERIVS, ADAMS, ADMNTP
     CALL ESPII (DERIVS, ADAMS, ADMNTP)
     END
     [User supplied subroutines]
*DERIVS
     [Equations defining derivatives and switches]
*ENDDERIVS
*ØUTPUT
     [Equations defining outputs]
     If (PRINT(1) .NE. 5HPRINT) GO TO 5
     WRITE(20) T, Y(1), Y(2), Y(3), ØMEGA, THETA
5    CONTINUE
     [More equations defining output, *PRINT, etc.]
*ENDØUT
     .
     .
     .
```

Notice that the MAIN program, here supplied by the user, is identical to the one normally written by PRECØMP (see Appendix D-3-b) except for the addition of TAPE20 to the PRØGRAM card, and that the MAIN program precedes all user subroutines and the *DERIVS section. Also notice that the WRITE statement is placed inside the output routine (Do not place it in the derivatives section!) and, in this case, that it will be executed only when ØUTPUT is called for print purposes, which will be at the print intervals specified on the *RUN card and at switch times if NDISPR > 0. This example will produce a file, TAPE20, with six values per record, one record for each print time, which will exist on magnetic tape after run termination and which may be read into another program for input or plotting by an unformatted READ statement.

VII. INPUTS

SECTION VII

INPUTS

The inputs needed for an ESP program may include any or all of the following: (1) the number of derivatives, (2) the starting and ending values for the independent variable T, (3) the print interval(s) desired, (4) the initial conditions (initial values) for the Y's, (5) variables which determine solution accuracy, (6) miscellaneous inputs which influence program control and format, and (7) any user variables which may be needed to compute initial conditions, derivatives, or switching characteristics. The number of inputs required and the manner in which they are supplied will depend upon the complexity of the user's problem and the degree of flexibility he wishes to build into the program.

A. NUMBER OF DERIVATIVES, START/STOP TIMES, AND PRINT INTERVALS (*RUN)

The only inputs required by every ESP job are the exact number of derivatives, the initial and final values of the independent variable and the printing interval(s). These are specified on the *RUN card, which is required for every run and is always the last card except for *GRAPH, *STØP or *RETURN (refer to Section VII-G-3). Notice that the print interval may be changed several times during the run, and that $t_{f_i} \geq t_o$. (Refer to Appendix F-1-e for directions on running the solution backward.) The format is

$$\boxed{\text{*RUN neq } t_o \text{ hprint}_1\ t_{f_1}\ \text{hprint}_2\ t_{f_2}\ \ldots .\ \$}$$

where

neq is an integer constant specifying the number of dependent variables (Y_i). It should equal the largest subscript of DY and must be corrected if derivatives are added or deleted.

t_o is a constant specifying the initial value of the independent variable (T) (typically 0.)

h_{print_i} is the printing interval to be used until the independent variable (T) reaches t_{f_i}

t_{f_i} is either the value of T at which the solution is to stop or, if it is followed by $hprint_n$ t_{f_n}, it is the time at which the print interval is to be changed.

$ is optional and terminates the information on the card

EXAMPLES:

1. *RUN 5 0. 0.5 10. $

   This runs the solution from 0. to 10.0, printing every 0.5 seconds, solving for 5 dependent variables (Y_i's).

2. *RUN 12 2.0 0.25 8.0 0.5 20.0 $

   This solves for 12 dependent variables, starting at 2.0 seconds, and printing every 0.25 second until T reaches 8.0 and then printing every 0.5 second until T reaches 20.0 seconds, at which time the solution stops.

B. INITIAL CONDITIONS: KNOWN CONSTANTS (*IV)

If the user wishes the initial values of all the dependent variables (Y_i's) to be zero, he does nothing. If he wishes any to be nonzero constants, the simplest way to input them is on the *IV card, which is placed among the run time data cards. IV's retain their value until they are reset on another *IV card or within the user's coding. The format is

*IV iv_1 iv_2 .... iv_n $

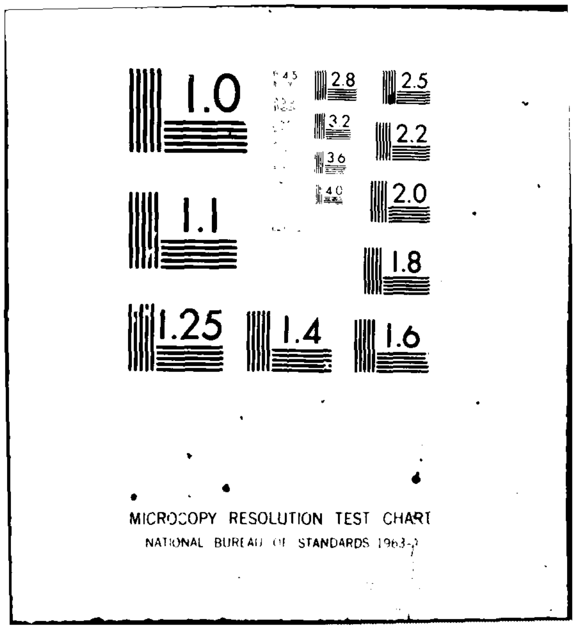
where

iv_i is either a constant alone or Yi = constant. Values are separated by blanks. If a constant appears alone, it is assumed to be the initial condition of the next dependent variable. If no value is given for a dependent variable, it is assumed to be zero.

$ is a required terminator

AD-A084 676 AEROSPACE CORP EL SEGUNDO CA ENGINEERING GROUP F/G 9/2
THE ECLECTIC SIMULATOR PROGRAM (ESP) USAGE GUIDE.(U)
MAY 80 E R COFFEY, H J WERTZ F04701-79-C-0080
UNCLASSIFIED TR-0080(9320)-1 SD-TR-80-21 NL

2 OF 3
AD
A084676


1.0
4.5
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

EXAMPLE:

```
*IV   0.   Y6=3.0   6.5   8.2   Y10=3.0   $

      produces:  Y(1)=Y(2)=Y(3)=Y(4)=Y(5)=0.0,
                 Y(6)=3.0,  Y(7)=6.5,  Y(8)=8.2,
                 Y(9)=0.0,  Y(10)=3.0 and any additional
                 Y(i)=0.0.
```

C. USER PARAMETERS: KNOWN CONSTANTS (*PAR)

An array named PAR of length 100 is created by ESP and automatically passed by the statement CØMMØN/PARS/PAR(100) to the derivative, output, switching, and initial computations routines for the convenience of the user. Its main purpose is to permit the user to introduce parameters into his equations without having to worry about how they will be defined or passed to each section, and to permit their values to be easily changed from run to run.

PAR values may be computed in ICCØMP (see below), but if known are most easily input to the PAR array on a *PAR card which, like the *IV card, is placed at the end of the program but before the *RUN to which it applies. PAR's retain their values until they are reset on another *PAR card or within the user's coding. They may be used at any time by name, i.e., PAR(3).

WARNING

PAR's must not be functions of T. Nonconstant PAR's used as inputs to SWTCH's or SWMEM's will cause errors in the determination of switching times, and use of PAR to pass nonconstant values to be printed will cause discrepancies in printed values.

The format is the same as for *IV

*PAR par_1 par_2 par_3 ... par_n $

where

par_i is either a constant alone or Pi = constant. Values are separated by blanks. If a constant appears alone, it is assumed to be the value of the next PAR(i). If no value is given for a PAR(i), it is assumed to be zero.

$ is a required terminator

EXAMPLES:

1.
```
*PAR  P2=6.75  4.0  P5=8.2  $
```

This gives:
```
PAR(1)=0.0
PAR(2)=6.75
PAR(3)=4.0
PAR(4)=0.0
PAR(5)=8.2
```

2. Use of the PAR array to change parameters easily in a series of program runs is illustrated in the following example:

If one of the derivative equations is

```
DY(3)=SIN(15.) * Y(1) + CØS(20.)*Y(2) + SIN(25.)*Y(3)
```

and it is desired to vary the angles over a range of values, the derivative equation can be written

```
DY(3)=SIN(PAR(1))*Y(1) + CØS(PAR(2))*Y(2) + SIN(PAR(3))*Y(3)
```

and the following *PAR cards used, one per run, without changing the equation

```
*PAR  15.0  20.0  25.0  $
*PAR  30.0  35.0  40.0  $
*PAR  72.0  75.0  78.0  $
```

*PAR and *IV cards are particularly useful in making multiple runs, that is, in making several runs of the same set of equations as one job, varying only the initial conditions, PAR's, or perhaps run times from one run to the next (see Appendix F-1, "Multiple Runs"). Note that since PAR's and IV's retain their values from run to run within a job until they are reset by the user, only those being changed for a particular run need to be redefined.

D. INITIAL CONDITIONS AND INPUTS TO BE COMPUTED (*ICCØMP...*ENDIC)

If any computation is necessary to set initial conditions or parameter values, the user will need to write an initial computations section into his coding. The start of this section is signaled by the card *ICCØMP, and the end is signaled by *ENDIC. Between these two cards, standard FØRTRAN and WHELP statements may be used, and their sequence is governed only by the usual rules of FØRTRAN and WHELP. All statements in this section will become part of the initial computations subroutine, ICCØMP, written by the ESP precompiler program, PRECØMP.

This subroutine is executed once (and only once) per run, so it is the proper place to perform any operation that is to be done only once before starting the solution, such as computing PAR's, computing initial values, reading in data, or rewinding tapes.

*ICCØMP.... *ENDIC may be used in addition to or in place of *PAR and *IV cards. Since the resulting subroutine is executed after *IV, *PAR, and *RUN have been encountered and processed, the user may input constants via *PAR or *IV and then safely use them on the right-hand side of expressions within *ICCØMP.

The program segment *ICCØMP...*ENDIC should be positioned after all user supplied routines, but before all run time cards such as *IV, *PAR, or *RUN (refer to Appendix A-3 for deck structure).

EXAMPLE:

```
              .
              .
              .
    *ICCØMP
            DIMENSIØN R(6)
            EQUIVALENCE(PAR(1), R), (PAR(7), RD), (PAR(8), DR)
            DATA PI/3.14159/
            RD=180./PI
            DR=PI/180.
    C INPUT  R ANGLES AS PARS IN DEGREES AND CØNVERT HERE.
            DO 5 I=1,6
            R(I)=DR*PAR(I)
        5   CØNTINUE
    *ENDIC
    *PAR  10.  20.  30.  40.  50.  60.  $
    *RUN  3  0.  0.5  10.0  $
```

E. DATA INPUT FROM CARDS OR USER FILES

Because an ESP job sometimes requires the input of data which cannot be conveniently handled by *PAR cards or DATA statements in ICCØMP, the user also has the option of reading it in with READ or NAMELIST statements. The data itself may either be on cards as part of the user's deck or it may be in the form of a user-created file (stored on magnetic tape or disc storage). In either case, the best place to read it from is within *ICCØMP... *ENDIC.

If the data exists on a manageable number of cards, the data cards may be placed immediately after the *RUN card(s) for the case(s) to which they apply. Since PRECØMP copies all cards beginning with the first run time control card onto TAPE 12, data cards so introduced will exist there during job execution along with the normal run time control cards. These data cards may be read as if they were on the standard INPUT file, e.g., READ 100, A, B, C.

NOTE

The user must take care to read exactly the correct number of cards since proper execution of *GRAPH or any other run time cards depends on the proper positioning of the input file. Also, input data may occupy only the first 72 columns, as columns 73-80 will not be copied to TAPE 12.

Sometimes, however, it may be more convenient to read input data from the user's own file. This would be true, for example, if the amount of data is very large, if it is necessary to test for end of file to terminate reading data for a case, or if t[illegible] user already has the data on a stored file of some sort. If this is the case, the user should:

- Create and save the data file (if it does not already exist), being sure to write in End-of-Files as he will need them.
- Add the proper job control cards to assign the saved file to the job and give it a logical name, say TAPEn.
- Write his own MAIN program (refer to Appendix D-3-b), adding TAPEn to the files declared on the PRØGRAM card.
- Read the data from the file in ICCØMP by using

```
        READ (n, føormat) list or        [ formatted read]
        READ (n, name)                   [namelist read]
        READ (n) list                    [unformatted read]
```

For more on data input used with stacked or multiple runs, refer to Appendix F.

F. INPUTS TO CONTROL ACCURACY

In addition to the inputs described above, there are several optional inputs which may be used to control the accuracy of the solution and the timing accuracy of discontinuities. All have default values but may be user-defined by means of special control cards placed in any order, among the run-time data cards, that is, after the *DERIVS, *ØUTPUT, and *ICCØMP sections, but before *RUN. *EPS and *Q control the solution accuracy directly and are described in Section IV-A-4. *HSW, *HSWM, and *HSWE are used to control the allowable timing error in SWTCH's, SWMEM's, and EVENT's, respectively, and are discussed in Section V-F.

G. MISCELLANEOUS INPUTS

Several other special input cards will be read and interpreted by PRECØMP. Their use is optional and they are provided mainly for the convenience of the user.

1. Print Headings

The print labels which appear on automatically formatted output are normally specified on the *PRINT card (refer to Section VI-A) from which they are read by PRECØMP and written onto a run-time data card called *HEADINGS, which the user will notice is printed with the other run time data cards at the beginning of his output. The user may, however, supply the *HEADINGS card himself, in which case his card completely supersedes the card written by PRECØMP. The number of labels on this card is the number of output variables which will be printed, even if it is less than the number of variables specified on the print card. (Thus a *HEADINGS card with no labels can be used to suppress printout.) The format is

*HEADINGS label_1 label_2 label_3 ... label_n $

where

*HEADINGS starts in column 1

label_i ≤ 10 hollerith characters (8 on IBM) with no embedded blanks, which will be used to label the ith output variable.

n is the number of variables that will be printed. It should be the same or less than the number of variables listed on the *PRINT card.

$ is a required terminator

2. Title for Printed Output

Placing the *TITLE card among the run-time data cards, somewhere before the *RUN card, causes whatever is in columns 8-71 to be used as the title printed at the top of every page of output. The format is

*TITLE [title]

3. Program Control

The following two cards may be used anywhere among the run-time data cards, to facilitate program control:

*RETURN

This causes program control to return to PRØGRAM MAIN at the point it is encountered, typically following a *RUN or *GRAPH command. It thereby permits execution of other statements in MAIN, such as calls to other subroutines or printing which is to be done only at the end of a run (see Section VI-C-5).

*STOP

This causes job termination at the point it is encountered. It should be used after *RUN if no *GRAPH follows and normal termination is desired at that point.

APPENDIX A

CARD FORMATS AND DECK STRUCTURE

CARD FORMATS AND DECK STRUCTURE

APPENDIX A

CARD FORMATS AND DECK STRUCTURE

A-1. GENERAL FORMAT RULES

General rules regarding ESP card formats are the following:

- All ESP "shorthand" cards begin with an asterisk (*) in column 1 and the first letter of the key word in column 2.
- Items on cards are separated either by blanks or by $ depending on their nature. (See specific cards.)
- Fixed length cards generally require no terminator, but those of variable length are terminated by a required $.
- FØRTRAN statements begin in column 7 and follow the usual rules of FØRTRAN.
- WHELP statements follow the usual format of WHELP (Appendix H).

General rules regarding use of comment cards:

- Comment cards are denoted only by a C in column 1. They may appear anywhere within:
  - user (sub)programs
  - the derivative computation section, delimited by *DERIVS and *ENDDERIVS
  - the output routine delimited by *ØUTPUT and *ENDØUT
  - the initial computation routine delimited by *ICCØMP and *ENDIC
  - the run-time data cards (provided they are <u>between</u> command cards).
- Comment cards may <u>not</u> be used:
  - within the data picked up in response to a command card, e.g., within the *PRINT specification.
  - between any routines, i.e., user routines, after *ENDDERIVS, *ENDØUT or *ENDIC, or after *PRINT when not within ØUTPUT.

A-2. SUMMARY OF CARD FORMATS

	Section Reference
● Defining derivatives and discontinuities:	
*DERIVS	III
*ENDDERIVS	
*BLOCK 1 β Y(i) e_{in} \$	III-C
*BLOCK 2 α_1 α_0 β_1 β_0 Y(i) Y(j) e_{in} \$	III-C
*SWTCH i 0_+ \$ 0_- \$ control_i \$	V-A
*SWMEM i input_i \$	V-B
● Defining output and initial computations:	
*ØUTPUT	VI-A-2
*ENDØUT	
*PRINT label_1=expression_1 \$... label_n=expression_n \$ \$	VI-A-1
*ICCØMP	VII-D
*ENDIC	
● Run-time data inputs:	
*IV iv_1 $iv_2 \ldots iv_n$ \$	VII-B
*PAR p_1 $p_2 \ldots p_n$ \$	VII-C
*SWITCHES n	V-D-1-b
*SWMEMCNT n	
*NEVENT n	V-C
*SWMEMSET n_1 $n_2 \ldots n_\ell$	V-B-4
*SWMEMDATA	V-B-3
i_1 c_1 $c_2 \ldots c_{10}$ \$	
i_2 c_1 $c_2 \ldots c_{10}$ \$	
⋮	
i_n c_1 $c_2 \ldots c_{10}$ \$	

	Section Reference
*MAXPLOTS n	VI-B
*RUN neq t_0 hprint_1 tf_1 hprint_2 tf_2 ... \$	VII-A
● Accuracy control:	
*EPS ϵ	IV-A-4
*Q q_1 q_2 ... q_n \$	IV-A-4
*HSW h_1 h_2 ... h_n \$	V-F
*HSWM h_1 h_2 ... h_n \$	V-F
*HSWE h_1 h_2 ... h_n \$	V-F
● Miscellaneous inputs:	
*TITLE----title----	VII-G-2
*HEADINGS label_1 label_2 ... label_n \$	VII-G-1
*STØP	VII-G-3
*RETURN	VII-G-3
*METHØD $\left\{\begin{array}{l}\text{RK2}\\\text{RK4}\\\text{PC}\end{array}\right.$	IV-A-1
● Plotting:	
*GRAPH n_x n_y [size][grid][scaling][type] [title] [X title] [Y title]	VI-B-2

A-3. USER'S DECK STRUCTURE

```
                      (CDC)                      (IBM)
*            [Job control cards  |       {  JCL cards
*       7-8-9 card               |       {  //SYSIN DD *

                          { RK2
             [*METHØD     { RK4
                          { PC

             [PRØGRAM MAIN (if written by user) *PROGRAM ON IBM

             [All user-supplied subroutines, including ICCØMP, ØUTPUT,
             [SWINPT, SWMEMN, EVENTS, and NØTIFY, if the user pro-
             [vides the entire routine. (See Appendix C-4 reserved names.)

*          ( [Coding which defines derivatives and discontinuities, in-
           ( [cluding all *SWTCH, *SWMEM, and *BLØCK statements
           ( [used, beginning with *DERIVS and ending with *ENDDERIVS.
*        ¶ { [Coding which defines the output: either *PRINT or
           ( [*ØUTPUT... *ENDØUT.
           ( [Coding which defines the initial computations: *ICCØMP...
           ( [*ENDIC.

             [Run-time data cards, in any order, including *IV, *PAR,
             [*SWITCHES, *SWMEMCNT, *SWMEMDATA, *SWMEMSET,
             [*EPS, *Q, *HSW, *HSWM, *HSWE, *NEVENT, *MAXPLØTS

*            [*RUN

             [Any input data to be read by a READ format, list or
              READ namelist.

             [*GRAPH

             [*STØP or *RETURN

                         |               {  /*
*       6-7-8-9          }               {  JCL cards
                         |               {  /*
```

[*] Only those sections or cards marked by an * are required. All others are optional.

[¶] The order of these three segments, *DERIVS, *ØUTPUT, and *ICCØMP, is interchangeable.

APPENDIX B

CONTROL CARDS AND FILE USAGE

APPENDIX B

CONTROL CARDS AND FILE USAGE

B-1. CDC CONTROL CARDS

(SCOPE 2.15 OPERATING SYSTEM)

> NOTE
>
> The following control card examples apply to the operating system in use at the publication date of this manual. Subsequent changes in operating system or control cards required will be documented as they occur.

ESP USED WITH WHELP

```
$PGMR....
$PARAM....
ATTACH, LIB1, 2NEWRESP.
ATTACH, LIB2, 3FTNPLØT LIB.
LIBRARY, LIB1, LIB2.
PRECOMP.
WHELP, IMSØRC.
FTN, I=TAPE15.
LGØ.
```

ESP USED WITHOUT WHELP

```
$PGMR....
$PARAM....
ATTACH, LIB1, 2NEWRESP.
ATTACH, LIB2, 3FTNPLØT LIB.
LIBRARY, LIB1, LIB2.
PRECØMP.
FTN, I=IMSØRC.
LGØ.
```

To get hard copy of film plots, add after LGØ.: HARDCPY.

B-2. IBM CONTROL CARDS (JCL)

(IBM 3033 MVS OPERATING SYSTEM)

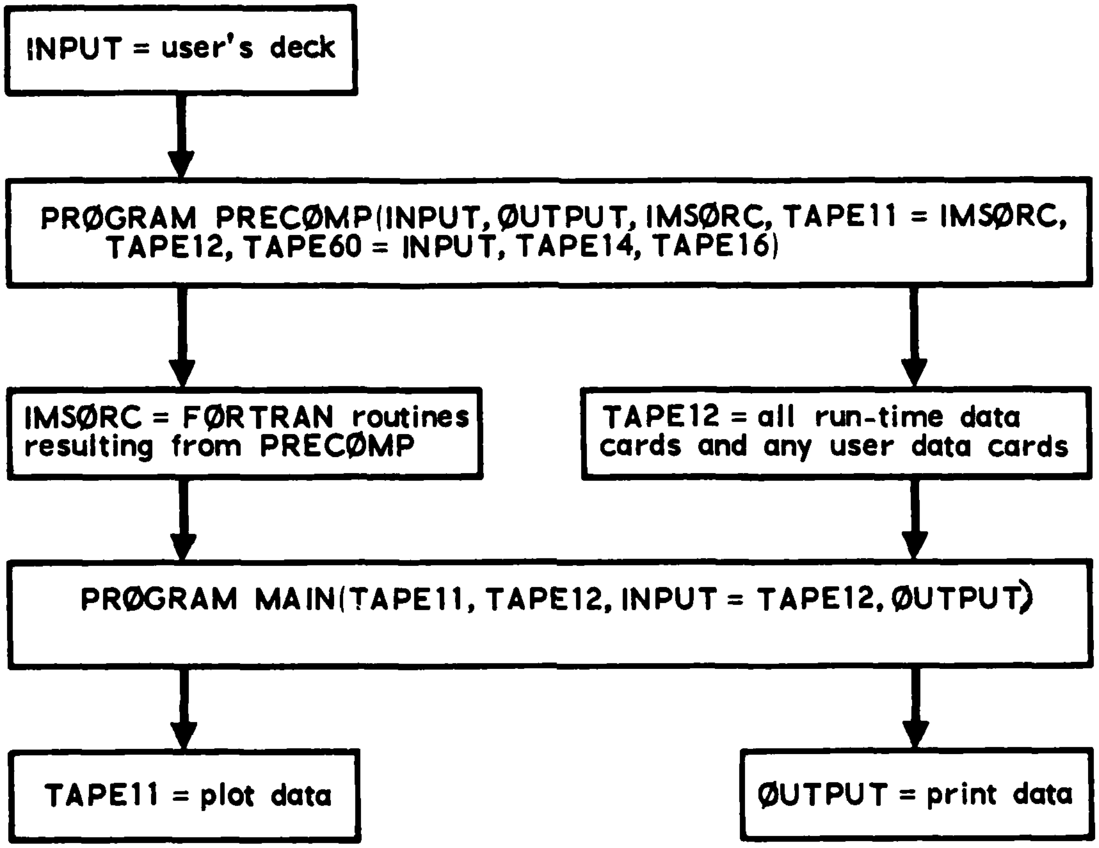
ESP WITH WHELP

```
//Z185  JØB...
//    MSGLEVEL...
/*JØBPARM ACCT...
//  EXEC   PGM=PRECØMP
//STEPLIB  DD    DSN=#4606.ESP.LIB(PRECØMP),DISP=SHR
//SYSPRINT  DD   SYSØUT=A
//TAPE11    DD    DSN=&TAPE11,DISP=(NEW,PASS),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//TAPE12    DD    DSN=&TAPE12,DISP=(NEW,PASS),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//TAPE14    DD    DSN=&&TAPE14,DISP=(NEW,DELETE),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//TAPE16    DD    DSN=&&TAPE16,DISP=(NEW,DELETE),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN   DD   DSN=#USERID.FILENAME.DATA,DISP-SHR
      (if user input resides on a permanent file)
                         or
//SYSIN   DD *
      (ESP source program cards)
//   EXEC   PGM=WHELP
//STEPLIB  DD  DSN=#4606.ESP.LIB(WHELP),DISP=SHR
//SYSPRINT    DD    SYSØUT=A
//SYSIN   DD   DSN=&TAPE11,DISP=(ØLD,DELETE)
//TAPE15   DD   DSN=&TAPE15,DISP=(NEW,PASS),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//TAPE11   DD   DSN=&TEMP,DISP=(NEW,DELETE),UNIT=VIØ,
//   SPACE=(TRK,(10,4)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//   EXEC   FØRTXCLG,CPARM='NØFØRMAT,AD(DBL),MAP',CØND.LKED=EVEN,
//        CØND.GØ=EVEN,LPARM=LET
//FØRT.SYSIN   DD   DSN=&TAPE15,DISP=(ØLD,DELETE)
//LKED.SYSLIB   DD   DSN=ØPUS.P077.SUBLIB,DISP=SHR
//GØ.FT11F001   DD   DSN=&&TAPE11,DISP=(NEW,DELETE),UNIT=VIØ,
//   SPACE=(TRK,(10,5)),DCB=(RECFM=VBS,BLKSIZE=6440,LRECL=16004)
//GØ.FT12F001   DD   DSN=&TAPE12,DISP=(ØLD,DELETE)
/*
```

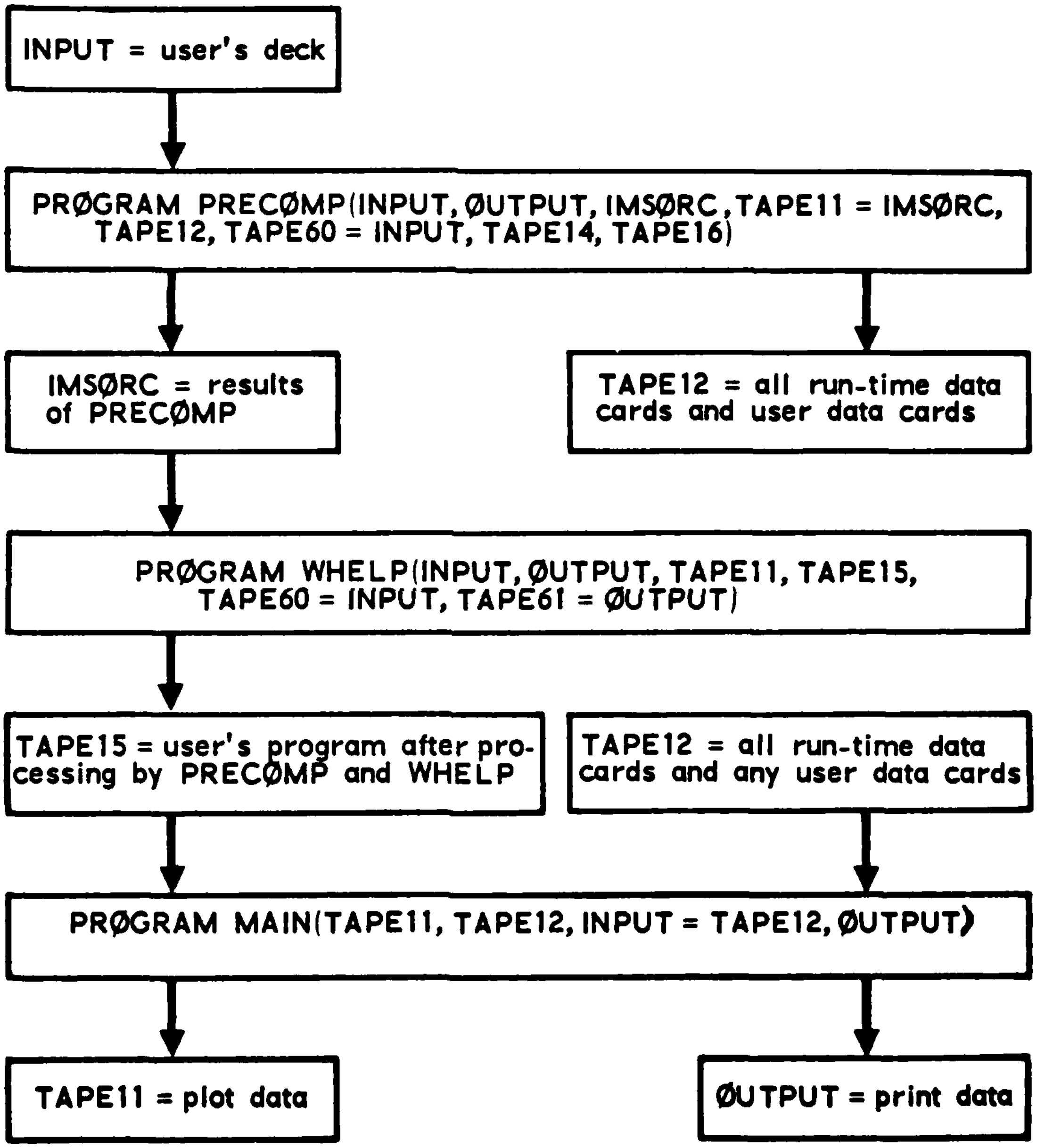
ESP WITHOUT WHELP

```
        ' Z185   JØB...
        ;,   MSGLEVEL...
        /*JØBPARM  ACCT...
00010   //   EXEC   PGM=PRECØMP
00020   //STEPLIB   DD   DSN=#4606.ESP.LIB(PRECØMP), DISP=SHR
00030   //SYSPRINT   DD SYSØUT=A
00040   //TAPE11    DD   DSN=&TAPE11, DISP=(NEW, PASS), UNIT=VIØ,
00050   //   SPACE=(TRK, (10, 5)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=800)
00060   //TAPE12    DD   DSN=&TAPE12, DISP=(NEW, PASS), UNIT=VIØ,
00070   //   SPACE=(TRK, (10, 5)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=800)
00080   //TAPE14    DD   DSN=&&TAPE14, DISP=(NEW, DELETE), UNIT=VIØ,
00090   //   SPACE=(TRK, (10, 5)), DCB=(RECFM=FB, LRECL=80), BLKSIZE=800)
00100   //TAPE16    DD   DSN=&&TAPE16, DISP=(NEW, DELETE), UNIT=VIØ,
00110   //   SPACE=(TRK, (10, 5)), DCB=(RECFM=FB, LRECL=80, BLKSIZE=800)
       { //SYSIN  DD  DSN=#USERID.FILENAME.DATA, DISP=SHR
       {     (if user input resides on a permanent file)
00120  {                     or
       { //SYSIN  DD  *
             (ESP source program cards)
00210   //   EXEC   FØRTXCLG, CPARM='NØFØRMAT, AD(DBL), MAP', CØND. LKED=EVEN,
00220   //       CØND.GØ=EVEN, LPARM=LET
00230   //FØRT.SYSIN   DD   DSN=&TAPE11, DISP=(ØLD, DELETE)
00240   //LKED.SYSLIB   DD   DSN=ØPUS.P077.SUBLIB, DISP=SHR
00250   //GØ.FT11F001   DD   DSN=&&TAPE11, DISP=(NEW, DELETE), UNIT=VIØ,
00260   //   SPACE=(TRK, (10, 5)), DCB=(RECFM=VBS, BLKSIZE=6440, LRECL=16004)
00270   //GØ.FT12F001   DD   DSN=&TAPE12, DISP=(ØLD, DELETE)
        /*
```

B-3. FILE USAGE FOR ESP WITHOUT WHELP


INPUT = user's deck
PRØGRAM PRECØMP(INPUT, ØUTPUT, IMSØRC, TAPE11 = IMSØRC, TAPE12, TAPE60 = INPUT, TAPE14, TAPE16)
IMSØRC = FØRTRAN routines resulting from PRECØMP
TAPE12 = all run-time data cards and any user data cards
PRØGRAM MAIN(TAPE11, TAPE12, INPUT = TAPE12, ØUTPUT)
TAPE11 = plot data
ØUTPUT = print data

B-4. FILE USAGE FOR ESP WITH WHELP


INPUT = user's deck
PRØGRAM PRECØMP(INPUT, ØUTPUT, IMSØRC, TAPE11 = IMSØRC, TAPE12, TAPE60 = INPUT, TAPE14, TAPE16)
IMSØRC = results of PRECØMP
TAPE12 = all run-time data cards and user data cards
PRØGRAM WHELP(INPUT, ØUTPUT, TAPE11, TAPE15, TAPE60 = INPUT, TAPE61 = ØUTPUT)
TAPE15 = user's program after processing by PRECØMP and WHELP
TAPE12 = all run-time data cards and any user data cards
PRØGRAM MAIN(TAPE11, TAPE12, INPUT = TAPE12, ØUTPUT)
TAPE11 = plot data
ØUTPUT = print data

APPENDIX C

PROGRAM VARIABLES AND RESERVED NAMES

APPENDIX C
PROGRAM VARIABLES
AND RESERVED NAMES

APPENDIX C

PROGRAM VARIABLES AND RESERVED NAMES

C-1. VARIABLES PASSED THROUGH CALLING SEQUENCES

Variable		Routines to which it is passed:
DY(100)	The derivative array	DERIVS ØUTPUT
IEVENT	An integer indicating the number of the EVENT being reported to SUBRØUTINE NØTIFY	NØTIFY
PLØT(100)	An array for storing the current value(s) of the plotted variables (Equivalent to VPLØT in CØMMON BLØCK UNIP1)	ØUTPUT
PRINT(60)	An array for storing the current value(s) of the printed variables	ØUTPUT
STØP	A variable which stops the current run if nonzero (Equivalent to T ≥ TFINAL)	DERIVS ØUTPUT
T	The independent variable, usually time	DERIVS ØUTPUT ICCØMP SWMEMN SWINPT
VALUES(50)	An array for storing the inputs to SWTCHs, SWMEMs, or EVENTs	SWINPT SWMEMN EVENTS
Y(100)	The dependent variable array	DERIVS ØUTPUT ICCØMP SWINPT SWMEMN

C-2. VARIABLES PASSED THROUGH COMMON BLOCKS

Variable	Block Name	Variable	Block Name
BUFFER(80)	READIN	NDISPR	NDISPR
CØLCNT	READIN	NEQ	MISCEL
CØNSTS(50, 10)	SWHPAR	NEVENT	SWTCHS
DY(100)	BASIC	NFIRST	READIN
DY(100, 9)	BLANK	NHEAD	UNIP2
EPS	MISCEL	NLØCAL	UNIP1
FIRSTP	RKCØNT	NØPLØT	UNIP1
FIXSTP	STPCØN	NPAGE	UNIP2
H	STPCØN	NPØINT	UNIP1
HEAD(60)	UNIP2	NTAP11	HMAXMN
HHMAX	HMAXMN	NUMSTP	HMAXMN
HHMIN	HMAXMN	ØUT(60)	UNIP2
HMAX	STPCØN	PAR(100)	PARS
HMIN	STPCØN	PLØT(2000)	BLANK
HP	STPCØN	Q(100)	MISCEL
HSW(50)	MISCEL	STØP(1)	MISCEL
HSWE(50)	MISCEL	SCR1(200)	BLANK
HSWM(50)	MISCEL	SWDBUG	SWDBUG
IFØRM(3)	UNIP2	SWMEM(50, 4)	SWTCHS
ISWTYP	SWHPAR	SWSET(50)	SWHPAR
JLINE	UNIP2	SWTCH(50)	SWTCHS
JSTART	STFPAR	T0	BASIC
KFLAG	STFPAR	TERMCH	READIN
KSV	SWHPAR	TF	BASIC
LINES	UNIP2	TITLE(8)	UNIP2
MF	STFPAR	TØDAY	UNIP2
MAX	UNIP1	TP	BASIC
MAXCHR	READIN	VALEVS(50, 2)	MISCEL
MAXCØL	READIN	VALMEM(50, 2)	MISCEL

Variable	Block Name	Variable	Block Name
MAXDER	STFPAR	VALUES(50, 2)	MISCEL
MAXMEM	SWTCHS	VPLØT(100)	UNIP1
MAXSWS	SWTCHS	Y(100, 9)	BLANK
MXL	UNIP2	Y0(100)	BASIC
NALTER	SWHPAR	YPRNT(100)	BASIC
NCHNG	SWHPAR		

C-3. ALPHABETICAL LIST OF COMMON BLOCKS AND THEIR CONTENTS

> NOTE
>
> Blocks SWTCHS and PARS are written by PRECØMP as part of SUBRØUTINES DERIVS, ØUTPUT, ICCØMP, SWINPT, and SWMEMN. All other blocks must be included by the user if he wishes to reference them. Block lengths are given in octal words for CDC use and hexadecimal bytes for IBM use since these are the bases used to list block size on maps generated by the two computers. Decimal size of each block can be easily obtained from the dimensions given with each variable name.

		Block Length	
		CDC (Octal	IBM (Hex)
CØMMØN/BASIC/T0, TF, TP, Y0(100), YPRNT(100), DY(100)		457	978
T0	The initial value of the independent variable	1	8
TF	The current final value of the independent variable (May be changed by the user's program)	1	8
TP	The last print time	1	8

		Block Length	
		CDC (Octal)	IBM (Hex)
Y0(100)	The initial conditions to be used when (next) *RUN is encountered. Set = 0 in ESP.	144	320
YPRNT(100)	The value(s) of the independent variables at the last print time. After a run this contains the "final" values of the Y's.	144	320
DY(100)	The value(s) of the derivatives at the last print time. After a run this contains the "final" values of the DY's.	144	320
CØMMØN/BLANK/PLØT(2000), SCR1(200), Y(100, 9), DY(100, 9)[¶]		7640	7D00
PLØT(2000)	Plot buffer	3720	3E80
SCR1(200)	Used internally as working space	310	640
Y(100, 9)	Y array, including past values of Y's	1604	1C20
DY(100, 9)	DY array, including past values of DY's	1604	1C20
CØMMØN/HMAXMN/HHMAX, HHMIN, NUMSTP, NTAP11		4	18
HHMAX	The maximum stepsize used thus far in the run. Automatically printed at end of run. May be tested or printed by user, but not changed.	1	8
HHMIN	The minimum stepsize used thus far in the run. Automatically printed at end of run. May be tested or printed by user but not changed.	1	8
NUMSTP	The number of integration steps taken thus far in the run. Automatically printed at end of run. May be tested or printed by user but not changed.	1	4
NTAP11	The number of data frames written onto TAPE11 for plotting. It is automatically printed at the end of the run.	1	4

[¶] The storage in this common block is used differently by different subroutines, but this describes the most common and generally relevant use. The user may wish at times to know the contents of this block, but should not alter them.

		Block Length	
		CDC (Octal)	IBM (Hex)
CØMMØN/MISCEL/STØP(1), Q(100), EPS, HSW(50), HSWM(50), HSWE(50), VALUES(50, 2), VALMEM(50, 2), VALEVS(50, 2), NEQ		1051	1144
STØP(1)	A variable which stops the current run if nonzero (equivalent to T ≥ TFINAL)	1	8
Q(100)	Q(i) is used to compute a maximum allowable absolute error in Y(i). It is set dynamically to MAX(Q(i), |Y(i)|).	144	320
EPS	EPS is used to compute relative error in Y(i): $$EPS \geq \sum \left(\frac{\text{error in } Y(i)}{Q(i)}\right)^2 .$$	1	8
HSW(50)	HSW(i) is the allowable timing error in determining SWTCH(i), normally set on the *HSW data card.	62	190
HSWM(50)	HSWM(i) is the allowable timing error in determining SWMEM(i), normally set on the *HSWM data card.	62	190
HSWE(50)	HSWE(i) is the allowable timing error in determining EVENT(i), normally set on the *HSWE data card.	62	190
VALUES(50, 2)	VALUES(i, 1) and VALUES(i, 2) store the current and previous values of the inputs to *SWTCH(i), alternately.	144	320
VALMEM(50, 2)	VALMEM(i, 1) and VALMEM(i, 2) store the current and previous values of the inputs to *SWMEM(i), alternately.	144	320
VALEVS(50, 2)	VALEVS(i, 1) and VALEVS(i, 2) store the current and previous values determining EVENT(i), alternately.	144	320
NEQ	The number of derivative equations to be integrated: set by the user on the *RUN card.	1	4

		Block Length	
		CDC (Octal)	IBM (Hex)
C∅MM∅N/NDISPR/NDISPR		1	4
NDISPR	A flag which controls printing at switching points. If 0, no print at switch times; if 1, one print at switch times; if 2, print and plotting occurs on left and right of each switch. (NDISPR = 1, nominally)	1	4
C∅MM∅N/PARS/PAR(100)		144	320
PAR(100)	An array for storing and automatically transmitting user variables, which may be easily input on a *PAR card.	144	320
C∅MM∅N/READIN/C∅LCNT, BUFFER(80), MAXC∅L, MAXCHR, NFIRST, TERMCH		125	158
C∅LCNT	Used internally by READIT: points tc beginning of next scan.	1	4
BUFFER(80)	Used internally by READIT: the current card in 80A1 F∅RMAT.	120	140
MAXC∅L	Used internally by READIT: the last column to be scanned.	1	4
MAXCHR	Used internally by READIT: maximum number of characters to be picked up in a hollerith field	1	4
NFIRST	Used internally by READIT: points to beginning of field just read.	1	4
TERMCH	Used internally by READIT: any hollerith character to mark the end of a field.	1	8
C∅MM∅N/RKC∅NT/FIRSTP		1	8
FIRSTP	A flag set to 1.0 by routine ESPCTL if Runge-Kutta is used to indicate the beginning of each step in fixed step mode or the beginning of each pair of steps in the variable step mode. Otherwise, FIRSTP=0.	1	8

		Block Length	
		CDC (Octal)	IBM (Hex)
CØMMØN/STFPAR/MF, KFLAG, JSTART, MAXDER		4	10
MF	Used internally	1	4
KFLAG	A flag returned from the integration routines to indicate success or failure of the integration step just taken. KFLAG=1 indicates error exceeded bounds and a warning message willbe printed.	1	4
JSTART	A flag used to indicate the start (restart) or continuation of integration. JSTART=0 when integration is starting or restarting. JSTART=1 when integration is continuing on from previous steps.	1	4
MAXDER	Used internally	1	4
CØMMØN/STPCØN/HP, H, FIXSTP, HMIN, HMAX		5	28
HP	The current printing interval. This is normally changed from the *RUN card but may be changed by the user's program during the run and must be > 0.	1	8
H	The current integration stepsize. If set ≠ 0 in ICCØMP this H will be tried first.	1	8
FIXSTP	The actual stepsize selected by the user for fixed stepsize integration using all Runge-Kutta	1	8
HMIN	A lower limit on the stepsize, nominally 0. May be set by user.	1	8
HMAX	The maximum stepsize permitted, nominally 1.0E50. May be set by user.	1	8

		Block Length	
		CDC (Octal)	IBM (Hex)
CØMMØN/SWDBUG/SWDBUG		1	4
SWDBUG	Logical variable which controls printing of data for switch debugging. If SWDBUG = .FALSE. (default) no print. If SWDBUG = .TRUE. print data to aid in debugging of switches.	1	4
CØMMØN/SWHPAR/NCHNG, NALTER, ISWTYP, KSV, CØNSTS(50, 10), SWSET(50)		1052	1078
NCHNG	A flag indicating whether any switches have just changed state during an integration step	1	4
NALTER	A flag used internally by SWTCHE	1	4
ISWTYP	Used internally	1	4
KSV	Used internally	1	4
CØNSTS(50, 10)	CØNSTS(i, j) is the constant Cj for SWMEMi. Although CØNSTS are normally defined on the *SWMEMDATA card, the user may include common block SWHPAR and define the CONSTS in ICCØMP. (No error test is made on CØNSTS so defined.)	764	FA0
SWSET(50)	The vector of values used to initialize SWMEMS in saturation rather than deadband. Normally input on the *SWMEMSET card.	62	C8
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT		375	7DC
SWTCH(50)	The magnitude is 1+ the number of times SWTCHi has switched. The sign is the current sign of the input. On the first call to DERIVS following a switching, all switches which have changed state have their magnitudes increased by 0.5.	62	190

		Block Length	
		CDC (Octal)	IBM (Hex)
SWMEM(50,4)	The output characteristics of SWMEM nonlinearities. SWMi=SWMEM(i, 3) -SWMEM(i, 2)*(SWMEM(i, 1)-"input"). SWMEM(i, 4) is a flag indicating the state and a change of state in SWMEMi.	310	640
MAXSWS	The maximum i for which SWCHi is serviced	1	4
MAXMEM	The maximum i for which SWMi is serviced	1	4
NEVENT	The number of EVENTs to be serviced, set by the user on the *NEVENT card.	1	4
<u>CØMMØN/UNIP1/VPLØT(100), NØPLØT, NPØINT, NLØCAL, MAX</u>		150	330
VPLØT(100)	Temporary storage for the current (100) PLØT variables	144	320
NØPLØT	A flag to prevent saving of any PLØT variables. If NØPLØT≠0, no PLØT variables are saved and ØUTPUT is only called at print times. (NØPLØT=0 nominally)	1	4
NPØINT	During run time, the actual number of points saved on TAPE11 for plotting. At end of run, the actual number of points to be plotted.	1	4
NLØCAL	During run time, the number of points in the PLØT buffer. (NLØCAL ≤ 2000 for CDC, 4000 for IBM.) At conclusion of run, the number of frames per plot point. Must <u>not</u> be changed by user.	1	4
MAX	The number of words per plot frame written onto TAPE11 by ESP	1	4

		Block Length	
		CDC (Octal)	IBM (Hex)
CØMMØN/UNIP2/HEAD(60), ØUT(60), TITLE(8), TØDAY, NHEAD, LINES, NPAGE, JLINE, MXL, IFØRM(3)		211	428
HEAD(60)	Vector which contains print headings normally picked up from *PRINT statement, but may be set directly by user-written FØRTRAN statements, or on *HEADINGS card.	74	1E0
ØUT(60)	Vector containing output values to be printed. Equivalent to PRINT(60) in ØUTPUT.	74	1E0
TITLE(8)	Vector containing title specified by user on *TITLE card	10	40
TØDAY	Contains actual date returned by subroutine DATE and printed on output.	1	8
NHEAD	The number of print variables (headings)	1	4
LINES	The number of print lines per block of print	1	4
NPAGE	Page number for printout	1	4
JLINE	Used internally by UNIP2 to control printed output	1	4
MXL	Used internally by UNIP2 to control printed output: number of blocks of printout per page.	1	4
IFØRM(3)	Contains output format to be used for printed output, based on accuracy requirements.	3	12

C-4. RESERVED SUBROUTINE NAMES

NOTE

The subroutines listed below are loaded and used during execution of an ESP job. The user should be careful not to duplicate any of these names when adding his own subroutines, except in the case of DERIVS, ICCØMP, ØUTPUT, SWINPT SWMEMN or MAIN when he intends to supply the entire routine himself.

ROUTINES USED BY ESP AND GRAPH

ABORT
ADAMS
ADMNTP
AND
BUFF
CKBLNK
CØMPL
CØNS
DECØD
DERIVS
EOFSIM
ENCØD
EØF
ESPCTL
ESPII
ESPRNT
ESPLØT
EVENTS
FILBUF
FILLBUF
FRAMES
FRAMXX
GENGRD
GRAPH
GRAPH2
GRAPHX
ICCØMP
ICKBLNK
IDECØD
IDFRAM
IPICK
JUNK
LABCHK
LIBRST
LIBSET
LIERR
LEVEL
LEVEL1
LEVEL2
LINGRD
LØGGRD
MAIN
NABLE
NEXTCHR
NEWGRD
NØTIFY
NUMBER
NUPLØT
NXTCHR
ØR
ØUTPUT
PACKER
PARRAY
PINØUT
PLØTS
PLTSYM
READIT
REMARK
RESTØR
SCALEP
SCALEPR
SECNZR
SHIFT
SKIPFIL
SKPFIL
STDGRD
SWINIT
SWINPT
SWMEMN
SWTCHE
SYMBØL
SYSTEM=
TIM2GØ
TIMEIN
TIMEØU
TIMEØUT

ROUTINES USED BY WHELP

CRØSS
IDENT
MATADD
MATINV
MATMAT
MATSUB
MATZRØ
MØVE
NEGATE
SCAMAT
TRNSML
TRNSPS

COMMON BLOCK NAMES (may not be used as subroutine names on IBM)

BASIC
BLANK
CØNSTS
E0FSIM
GRAPHP
HMAXMN
LIBSCR
MISCEL
NDISPR
PARS
READIN
RKCØNT
STFPAR
STPCØN
SWDBUG
SWHPAR
SWTCHS
TEMSTR
UNIP1
UNIP2

APPENDIX D

PROGRAM CONTROL AND EXECUTION

APPENDIX D

PROGRAM CONTROL AND EXECUTION

D-1. INTRODUCTION

How an ESP program works can be considered on two levels. First, there is the manner in which the control cards put the program together from the user's deck and the ESP files. Then, there is the manner in which the program actually executes to solve the user's problem. This appendix will attempt to clarify both, first by providing a diagram showing the relationship of control cards, compilers, libraries and files and second by providing descriptions and flowcharts of the major subroutines which make up the ESP library.

D-2. ESP CONTROL CARDS AND WHAT THEY DO

D-2-a. CDC

CDC	INPUT	ACTION	OUTPUT
PRECØMP.	(User's coding)	Executes program PRECØMP, which reads the user's code and translates it to FØRTRAN, adding statements as needed to make complete routines.	➤IMSØRC, a file containing user routines, MAIN, ICCØMP, ØUTPUT, DERIVS, SWINPT, and SWMEMN ➤TAPE12, a file containing the run-time data cards and user data cards
WHELP(IMSØRC) (optional)	IMSØRC	Executes program WHELP, a FØRTRAN program which reads and translates vector matrix equations written in WHELP language into FØRTRAN.	➤TAPE15, a file containing user's program in FØRTRAN

FTN(I=IMSØRC)
or
FTN(I=TAPE15)
	IMSØRC (if no WHELP) or TAPE15 (if WHELP used)	Compiles the input file into executable binary code.	➤LGØ, a relocatable binary version of the user's program

LGØ.	LGØ TAPE12	Loads the program, adding the run-time routines from the ESP library. Executes program MAIN, using TAPE12 as an input file.	➤TAPE11, containing plot data, packed. ➤ØUTPUT, containing print data.

D-2-b. IBM

IBM-JCL	INPUT	ACTION	OUTPUT
Cards 010 to 120*	(User's coding)	Execute program PRECØMP, which reads the user's code and translates it to FØRTRAN, adding statements as needed to make complete routines.	➤ TAPE11, a file containing user routines, MAIN, ICCØMP, ØUTPUT, DERIVS, SWINPT, and SWMEMN. ➤ TAPE12, a file containing the run-time data cards and user data cards.
Cards 130 to 200* (optional)	TAPE11	Execute program WHELP, a PLI program which reads and translates vector matrix equations written in WHELP language into FØRTRAN.	➤ TAPE15, a file containing user's program in FØRTRAN.
Cards 210 to 270*	TAPE11 (if no WHELP) or TAPE15 (if WHELP used) TAPE12	Compile, link edit and execute program MAIN, adding the run-time routines from the ESP library and using TAPE12 as an input file.	➤ TAPE11, containing plot data, packed. ➤ SYSØUT, containing print data.

*See Appendix B, Section B-2 for a complete listing of these JCL cards.

D-3. PRECØMP

D-3-a. What PRECØMP Does

PRØGRAM PRECØMP is a precompiler, written in FØRTRAN on CDC and PLI on IBM, which reads the user's ESP language input deck or file and translates it into executable FØRTRAN routines¶ to be used by the ESP run-time package. Its chief functions are to "crack" the *control cards such as *BLØCK, *SWTCH, and *PRINT, and to write the additional cards needed to complete those subroutines based on user coding, namely MAIN, DERIVS, ICCØMP, ØUTPUT, SWINPT, and SWMEMN (see Appendix D-3-b).

PRECØMP operates by making repeated calls to SUBRØUTINE READIT (which reads the user's card images) and by writing these card images out onto file IMSØRC until a signal card is detected. It then tests the signal card to determine its next action, which may be copying more cards, setting flags, translating the data on the signal card, or writing additional FØRTRAN statements onto IMSØRC to complete the subroutines. Since *READIT depends upon* blanks, $ terminators, and * to delimit fields of data and to tell it how to handle data, formats for all ESP cards should be followed carefully.

Each time PRECØMP is executed, it will do the following ten steps in order, although substeps may be in any order, as indicated:

1. Call TIMEIN to get precompiler starting time.
2. Process *METHOD card if used.
3. Write PRØGRAM MAIN, specifying proper integration package, <u>or</u> copy user's PRØGRAM MAIN, if provided.
4. Copy all user-supplied subroutines and/or functions.

¶If WHELP statements are used, however, they are copied as is and must be converted to FØRTRAN by the WHELP precompiler.

5. Write SUBRØUTINES DERIVS, ICCØMP, and ØUTPUT in any order as follows:

   a. Write SUBRØUTINE ICCØMP using all cards contained between *ICCØMP and *ENDIC. If no *ICCØMP is used, write a dummy routine.

   b. Write SUBRØUTINE ØUTPUT using all cards contained between *ØUTPUT and *ENDØUT. If no *ØUTPUT appears, use data on *PRINT card. If neither *ØUTPUT nor *PRINT is used, write a dummy routine.

   c. Write SUBRØUTINE DERIVS using all cards contained between *DERIVS and *ENDDERIVS. Within this section, do the following in any order:

      (1) Copy FØRTRAN and WHELP statements as given.

      (2) Translate *BLØCK cards into FØRTRAN and write as part of DERIVS.

      (3) Process *SWTCH and *SWMEM cards by writing SUBRØUTINES SWINPT and SWMEMN containing the input expressions and by adding code to DERIVS to define SWTCH and SWMEM output.

6. Write *HEADINGS, *SWTCHES, and *SWMEMCNT cards on TAPE12.

7. Copy remaining input, such as *IV, *PAR, or *RUN cards and user data cards, up to EØF, onto TAPE12.

8. ENDFILE 12 and REWIND.

9. ENDFILE IMSØRC and REWIND.

10. Call TIMEØUT to compute precompiler time used.

D-3-b. CARDS WRITTEN BY PRECØMP

Below is a listing of the cards written by PRECØMP which are added to the various segments of the user's coding to produce complete subroutines. If the user chooses to provide any or all of these routines himself, he should be careful to include all of the cards listed and to insert his own coding where indicated. Notice that when a particular routine has no function and is written as a dummy (see SUBRØUTINE ICCØMP in Example Problem, Section II) not all of the cards below will be listed. Also, if the WHELP precompiler is used, it writes additional common block statements for its own needs.

MAIN PROGRAM

```
PRØGRAM MAIN (TAPE11, TAPE12, INPUT=TAPE12, ØUTPUT)
EXTERNAL DERIVS, METHØD, INTERP
CALL ESPII (DERIVS, METHØD, INTERP)
END
```

where

	Adams	RK2	RK4	Predictor/Corrector
METHØD=	ADAMS	ESPRK2	ESPRK4	ESPPC
INTERP=	ADMNTP	INRKPC	INRKPC	INRKPC

DERIVATIVE SUBROUTINE

```
SUBRØUTINE DERIVS(T, Y, DY, STØP)
DIMENSIØN Y(100), DY(100), PAR(100)
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT
CØMMØN/PARS/PAR
    [coding which defines derivative equations as DY's and the desired
    [outputs of any discontinuities used.
RETURN
END
```

*SWTCH INPUTS SUBROUTINE

```
SUBRØUTINE SWINPT (VALUES, T, Y)
DIMENSIØN VALUES(1), Y(1), PAR(100)
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT
CØMMØN/PARS/PAR
    [coding which defines input expressions to SWTCHs and stores them
    [in array VALUES.
RETURN
END
```

*SWMEM INPUTS SUBROUTINE

```
SUBRØUTINE SWMEMN (VALUES,T,Y)
DIMENSIØN VALUES(1), Y(1), PAR(100)
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT
CØMMØN/PARS/PAR
     [coding which defines input expressions to SWMEMs and stores them
     [in array VALUES.
RETURN
END
```

OUTPUT SUBROUTINE

```
SUBRØUTINE ØUTPUT (T,Y,DY,PLØT,PRINT,STØP)
DIMENSIØN Y(100), PAR(100), PLØT(10), PRINT(60), DY(100)
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT
CØMMØN/PARS/PAR
     [coding which defines print and plot values and stores them in
     [PRINT and PLØT, respectively.
RETURN
END
```

INITIAL COMPUTATIONS SUBROUTINE

```
SUBRØUTINE ICCØMP(T,Y)
DIMENSIØN Y(100), PAR(100)
CØMMØN/SWTCHS/SWTCH(50), SWMEM(50, 4), MAXSWS, MAXMEM, NEVENT
CØMMØN/PARS/PAR
     [coding which defines any initial conditions, computes program
     [constants, or performs any task involved with program initialization.
RETURN
END
```

D-4. RUN-TIME ROUTINES

Once PRECØMP is finished, IMSØRC will contain PRØGRAM MAIN, SUBRØUTINES ICCØMP, ØUTPUT, DERIVS, SWINPT, SWMEMN, EVENTS and NØTIFY, and any other subroutines provided by the user to his program. In order to complete the program and make it executable, a group of subroutines to be referred to as "run-time" routines will be selected from the ESP library and added to the program. The internal workings of most of these routines probably are not relevant to the user, but a brief description of each follows.

To aid the user in understanding ESP and perhaps in debugging his program, some further information is included. Appendix D-4-b shows the overall relationship of subroutines during execution. Further, Appendices D-4-c, i-vi, contain schematic flowcharts of those run-time routines most significant in program control and logic, namely, SUBRØUTINES ESPII and ESPCTL, and the integration routines ADAMS, ESPRK4 (ESPRK2), and ESPPC.

D-4-a. Routines Provided by ESP (Run-Time Routines)

ESPII — Controls overall execution by such operations as reading and interpreting run-time cards, controlling multiple cases, and calling plot routines (see flowchart, Appendix D-4-c-ii).

ESPCTL — Controls all of the tasks needed to execute one *RUN card, which includes initializing and printing variables, calling the integrator routine selected, printing warnings if integration was unsuccessful, checking for switches and calling the appropriate switch routines, and storing print and plot data at the correct times (see flowchart., Appendix D-4-c-iii).

ØDESØL — The integration routine, which will be one of the following:

ESPPC	Predictor-corrector method
ESPRK2	Second order Runge-Kutta
ESPRK4	Fourth order Runge-Kutta
ADAMS	Adams integration

(See Chapter IV, Integration Package, and Appendices D-4-c, iv, v, vi.)

INTERP — The interpolation routine used to interpolate data for printout and switchings, which will be one of the following:

INRKPC	Used for predictor-corrector, RK2 or RK4
ADMNTP	Used for Adams integration

ESPRNT — Calls ØUTPUT to obtain print data and does actual printing of output.

ESPLØT — Calls ØUTPUT to obtain plot data and stores plot data for later plotting.

SWINIT — Initializes switches and reinitializes them after a switching.

SWTCHE — Evaluates SWTCH, SWMEM, and EVENTS inputs by calling SWINPT, SWMEMN, and EVENTS, detects sign or state changes, locates zero crossings, and flags outputs.

SECNZR — Finds the zero crossing when SWTCHS, SWMEMs, or EVENTS have been detected.

DATE — Returns date on which run is executed.

READIT — Reads data from specified input stream, terminating on the character indicated (blank or $).

TIMEIN — Records solution starting time.

TIMØUT — Records solution stop time.

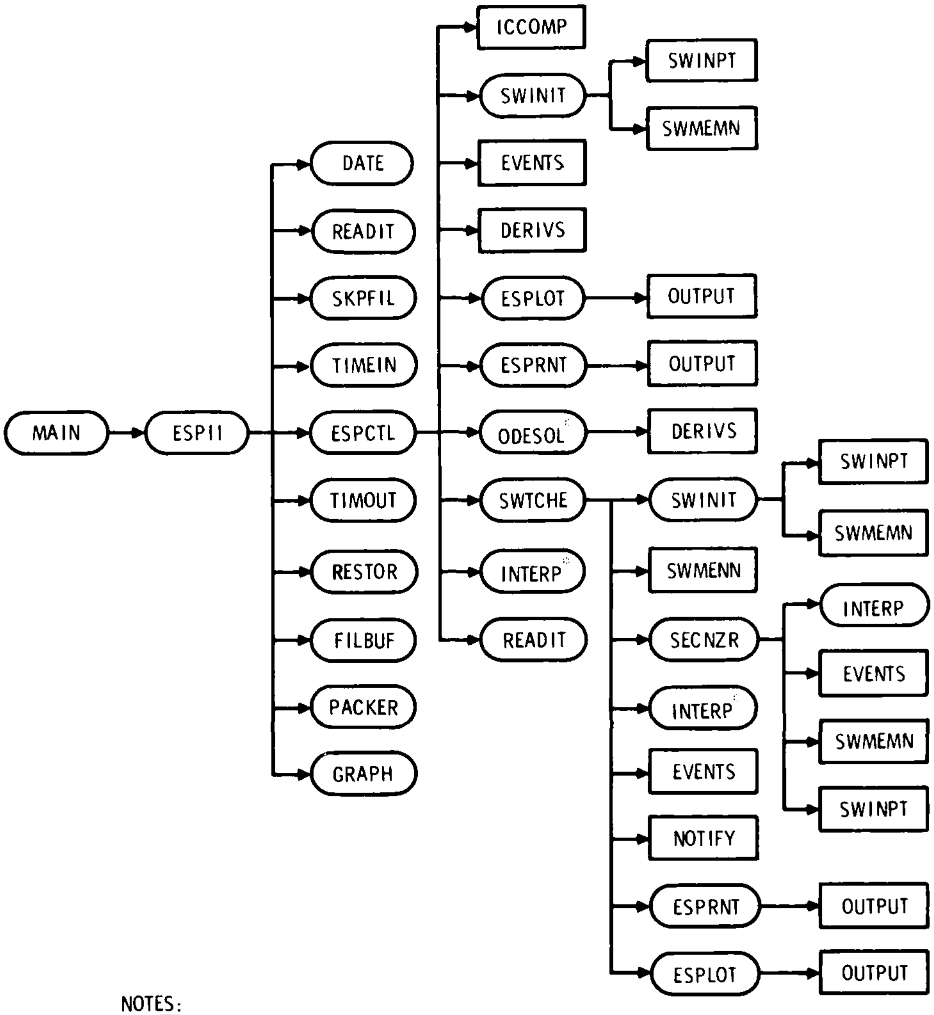
GRAPH — Does actual plotting.

RESTØR — Used to manipulate plot buffers.

FILBUF
PACKER
SKPFIL — Used internally for file reading and manipulation.

DIFTAB — Called by ESPPC to see if stepsize doubling will introduce numerical instability.

D-4-b. Relationship of Routines During Execution



NOTES:

1. (NAME) - routine is part of ESP

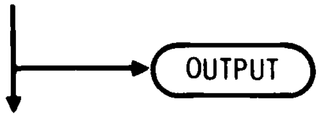
   □ NAME - routine contains user code

2. Many routines are called more than once by their calling routine but are shown only once
3. ODESOL* and INTERP* are internal variables representing the following subroutine names, depending on the integration method selected:

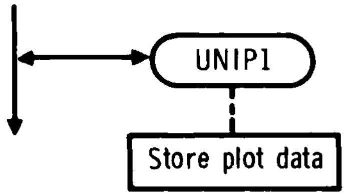
Method	ODESOL	INTERP
Predictor/Corrector	ESPPC	INRKPC
RK2	ESPRK2	INRKPC
RK4	ESPRK4	INRKPC
ADAMS	ADAMS	ADMNTP

D-4-c. Flowcharts of Significant Routines

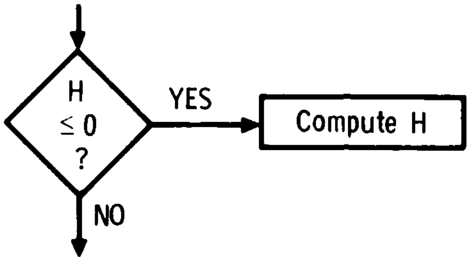
D-4-c-i. Explanation of Flowchart Conventions



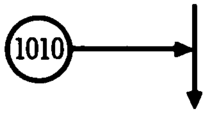
Call the subroutine whose FORTRAN name is OUTPUT



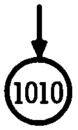
Call subroutine UNIP1, which performs the function described in the box beneath it. The two way arrow indicates that program control returns to the main line at completion of the subroutine



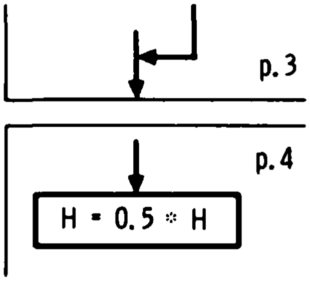
Compute H only if H ≤ 0; then continue on "NO" path



Enter at this point after a branch from another location



Branch from this point to the corresponding entry

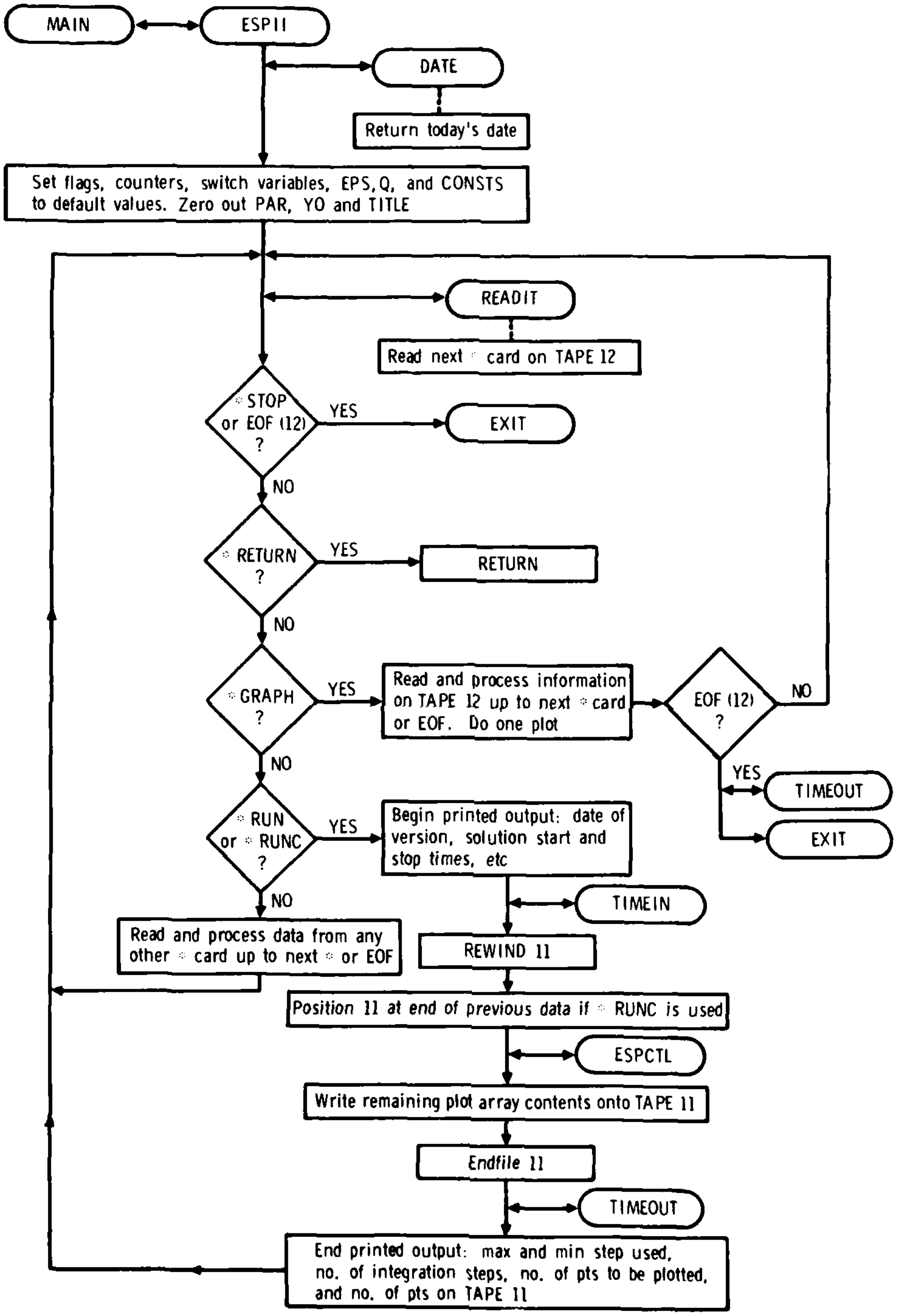


Multipage flowcharts continue from the dangling arrow in the lower left corner of a page to the entry arrow at the upper left of the next page

FIRSTP

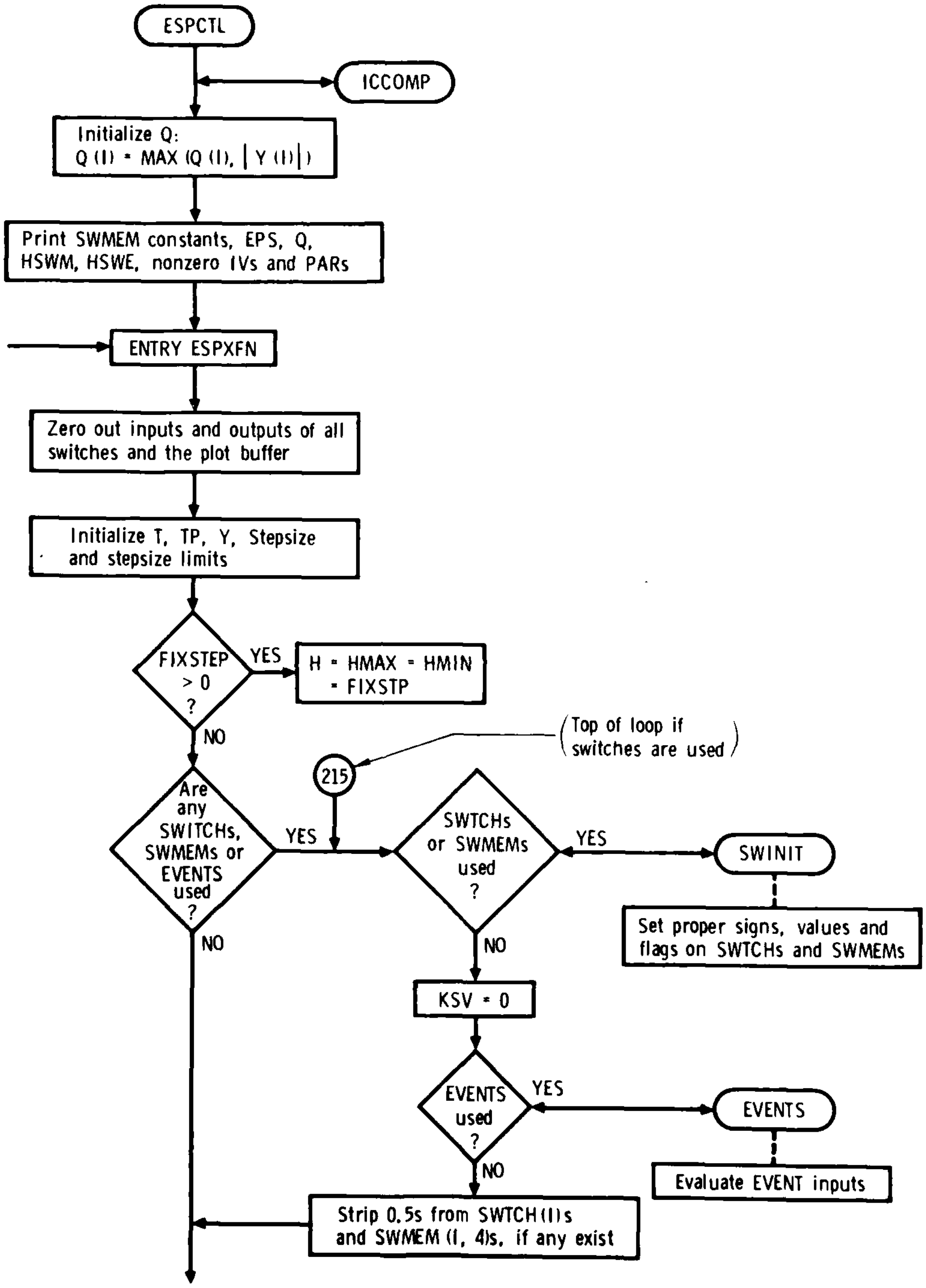
Names appearing in all capital letters represent actual FORTRAN names

D-4-c-ii. Flowchart of Subroutine ESPII (page 1 of 1)

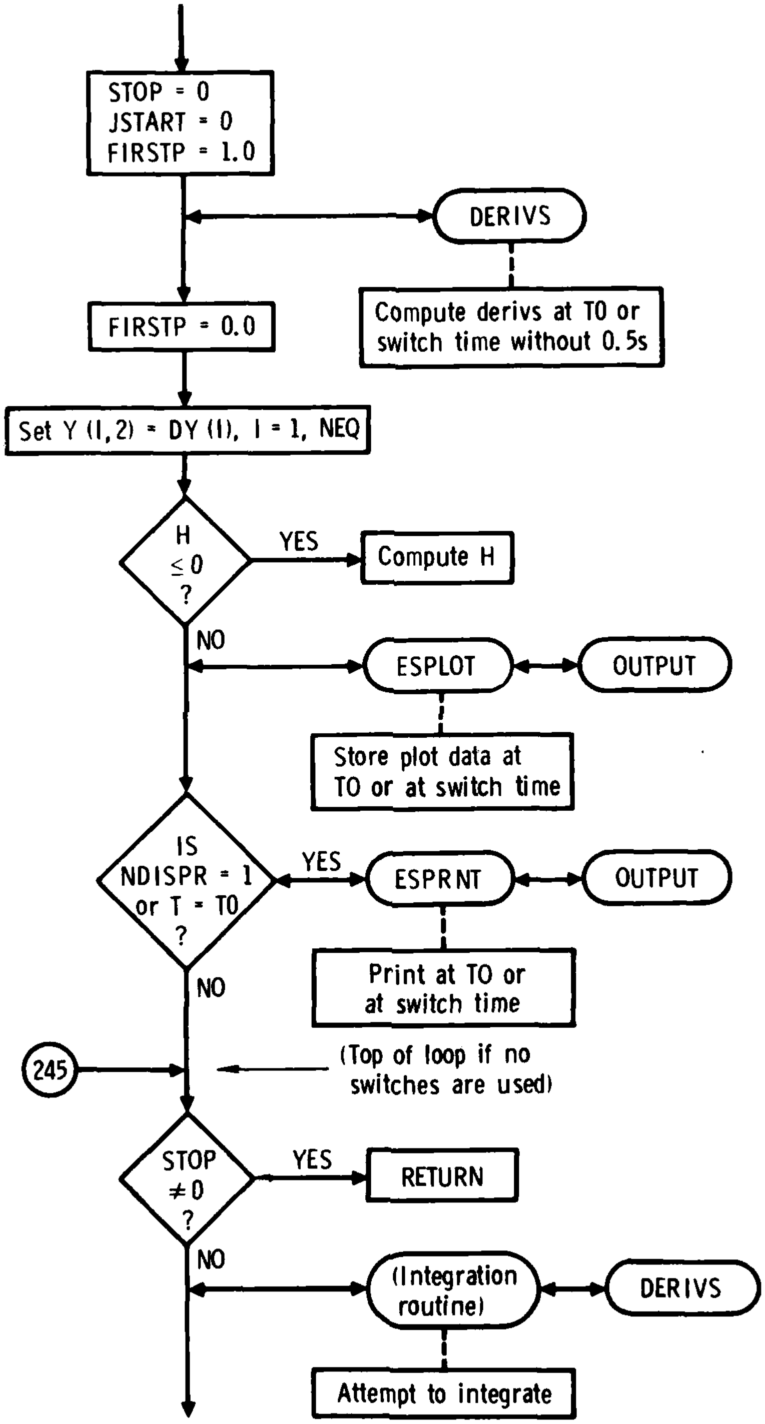


MAIN
ESPII
DATE
Return today's date
Set flags, counters, switch variables, EPS, Q, and CONSTS to default values. Zero out PAR, YO and TITLE
READIT
Read next * card on TAPE 12
* STOP or EOF (12) ?
YES
EXIT
NO
* RETURN ?
YES
RETURN
NO
* GRAPH ?
YES
Read and process information on TAPE 12 up to next * card or EOF. Do one plot
EOF (12) ?
NO
YES
TIMEOUT
EXIT
NO
* RUN or * RUNC ?
YES
Begin printed output: date of version, solution start and stop times, etc
NO
Read and process data from any other * card up to next * or EOF
TIMEIN
REWIND 11
Position 11 at end of previous data if * RUNC is used
ESPCTL
Write remaining plot array contents onto TAPE 11
Endfile 11
TIMEOUT
End printed output: max and min step used, no. of integration steps, no. of pts to be plotted, and no. of pts on TAPE 11

D-4-c-iii. Flowchart of Subroutine ESPCTL
(page 1 of 3)

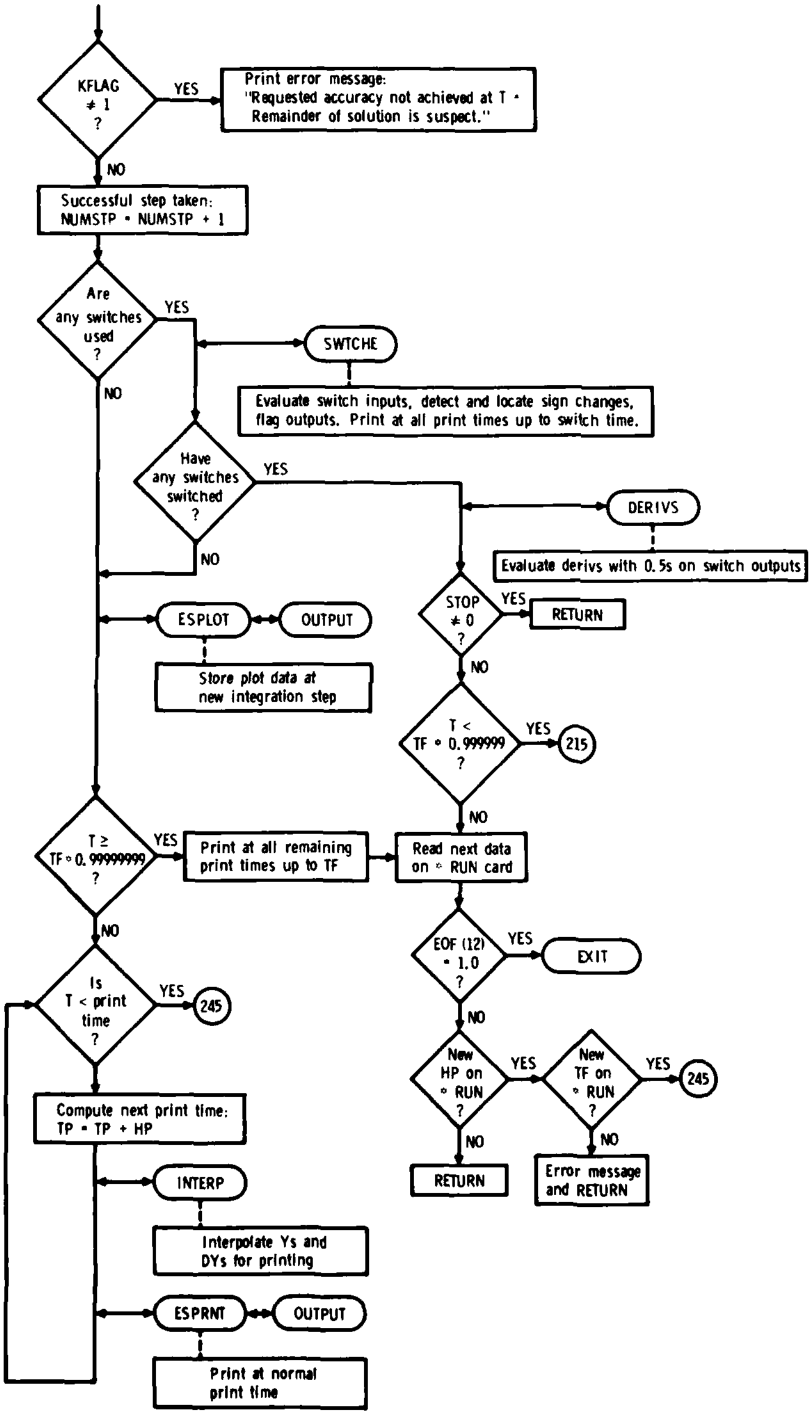

ESPCTL
ICCOMP
Initialize Q:
Q (I) = MAX (Q (I), | Y (I)|)
Print SWMEM constants, EPS, Q, HSWM, HSWE, nonzero IVs and PARs
ENTRY ESPXFN
Zero out inputs and outputs of all switches and the plot buffer
Initialize T, TP, Y, Stepsize and stepsize limits
FIXSTEP > 0 ?
YES
H = HMAX = HMIN = FIXSTP
NO
(Top of loop if switches are used)
215
Are any SWITCHs, SWMEMs or EVENTS used ?
YES
SWTCHs or SWMEMs used ?
YES
SWINIT
Set proper signs, values and flags on SWTCHs and SWMEMs
NO
NO
KSV = 0
EVENTS used ?
YES
EVENTS
Evaluate EVENT inputs
NO
Strip 0.5s from SWTCH(I)s and SWMEM (I, 4)s, if any exist

D-4-c-iii. Flowchart of Subroutine ESPCTL (page 2 of 3)

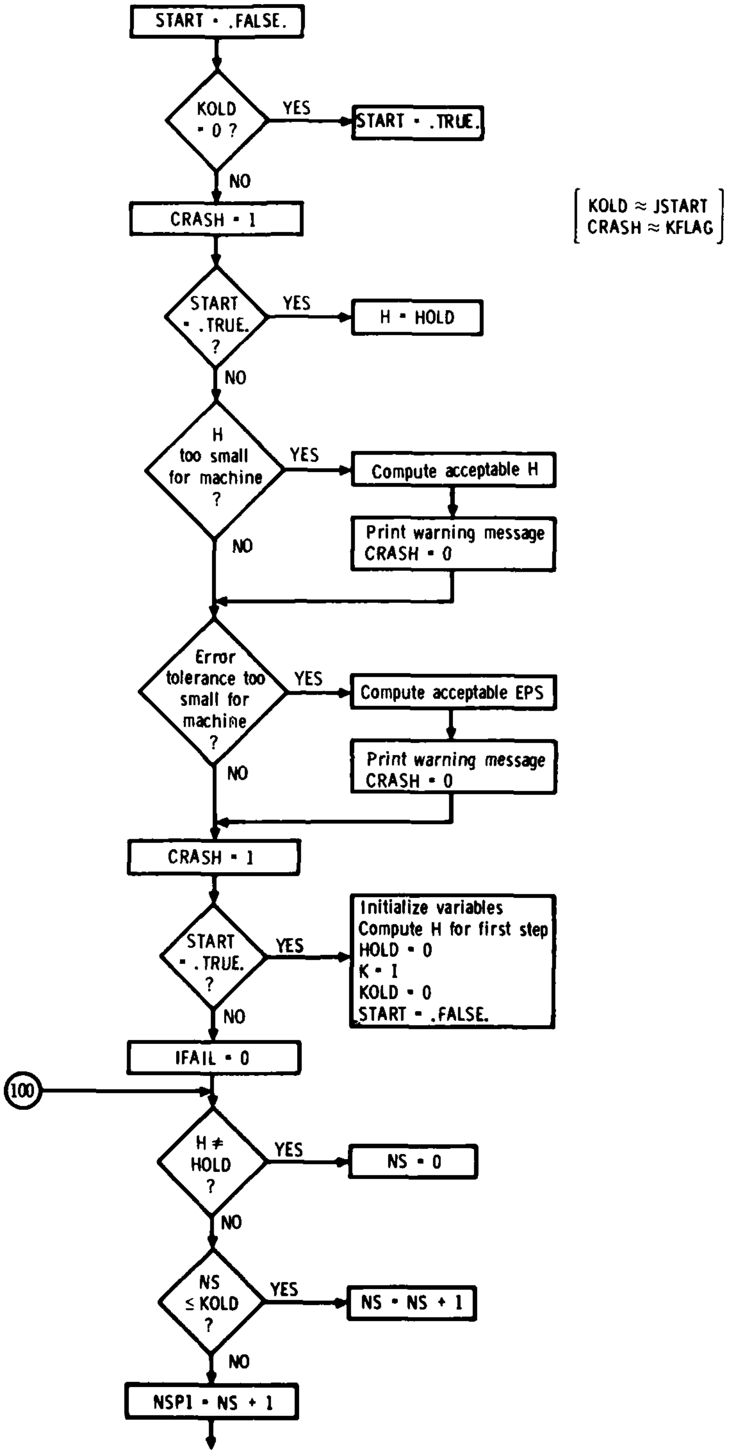


STOP = 0
JSTART = 0
FIRSTP = 1.0
DERIVS
Compute derivs at T0 or switch time without 0.5s
FIRSTP = 0.0
Set Y (I,2) = DY (I), I = 1, NEQ
H ≤ 0 ?
YES
Compute H
NO
ESPLOT
OUTPUT
Store plot data at T0 or at switch time
IS NDISPR = 1 or T = T0 ?
YES
ESPRNT
OUTPUT
Print at T0 or at switch time
NO
245
(Top of loop if no switches are used)
STOP ≠ 0 ?
YES
RETURN
NO
(Integration routine)
DERIVS
Attempt to integrate

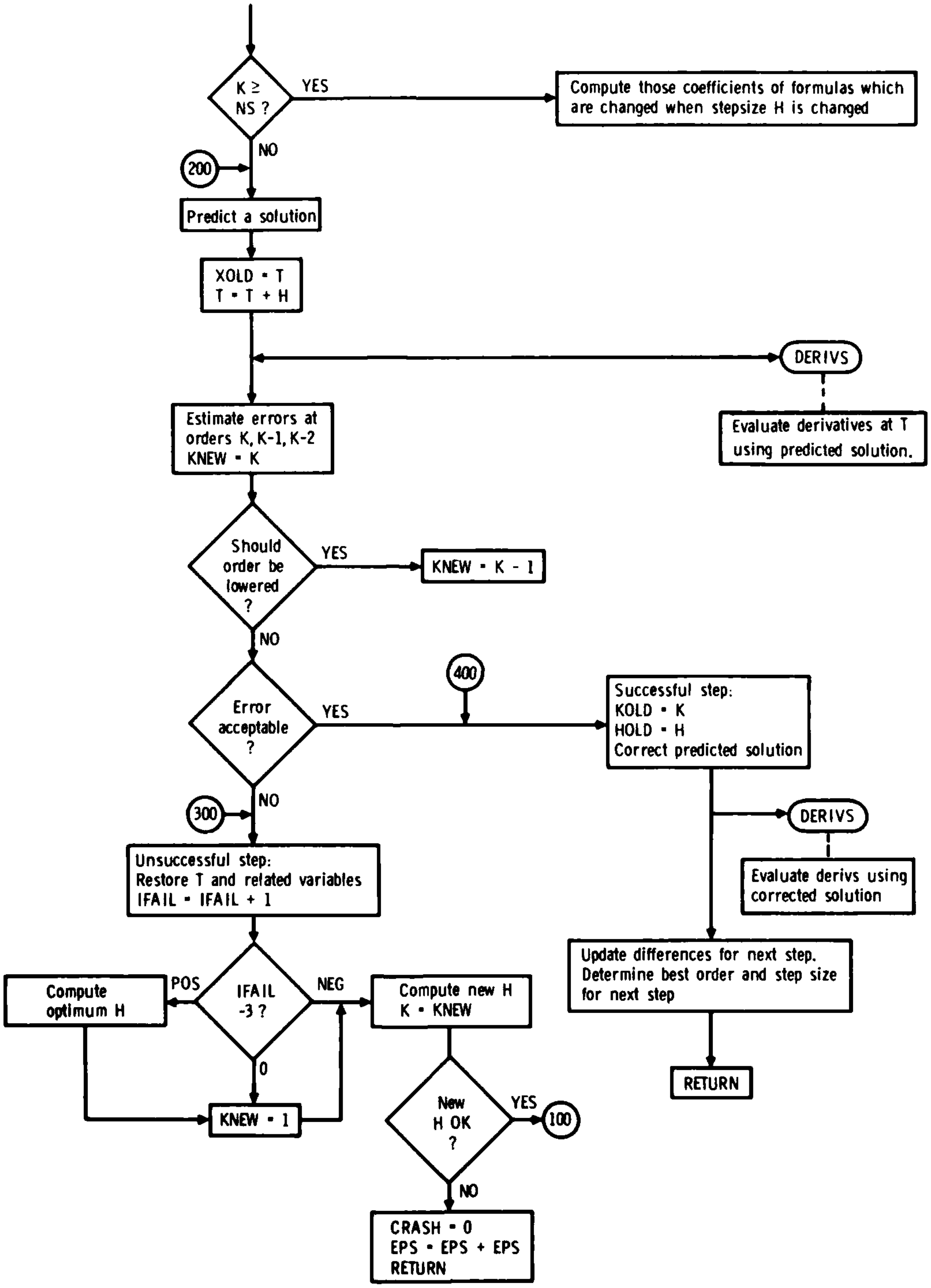
D-4-c-iii. Flowchart of Subroutine ESPCTL (page 3 of 3)


KFLAG ≠ 1 ?
YES
Print error message: "Requested accuracy not achieved at T = Remainder of solution is suspect."
NO
Successful step taken: NUMSTP = NUMSTP + 1
Are any switches used ?
YES
SWTCHE
Evaluate switch inputs, detect and locate sign changes, flag outputs. Print at all print times up to switch time.
NO
Have any switches switched ?
YES
DERIVS
Evaluate derivs with 0.5s on switch outputs
NO
ESPLOT
OUTPUT
Store plot data at new integration step
STOP ≠ 0 ?
YES
RETURN
NO
T < TF = 0.999999 ?
YES
215
NO
T ≥ TF = 0.99999999 ?
YES
Print at all remaining print times up to TF
Read next data on = RUN card
NO
EOF (12) = 1.0 ?
YES
EXIT
NO
Is T < print time ?
YES
245
New HP on = RUN ?
YES
New TF on = RUN ?
YES
245
NO
RETURN
NO
Error message and RETURN
Compute next print time: TP = TP + HP
INTERP
Interpolate Ys and DYs for printing
ESPRNT
OUTPUT
Print at normal print time

D-4-c-iv. Flowchart of Subroutine ADAMS
(page 1 of 2)

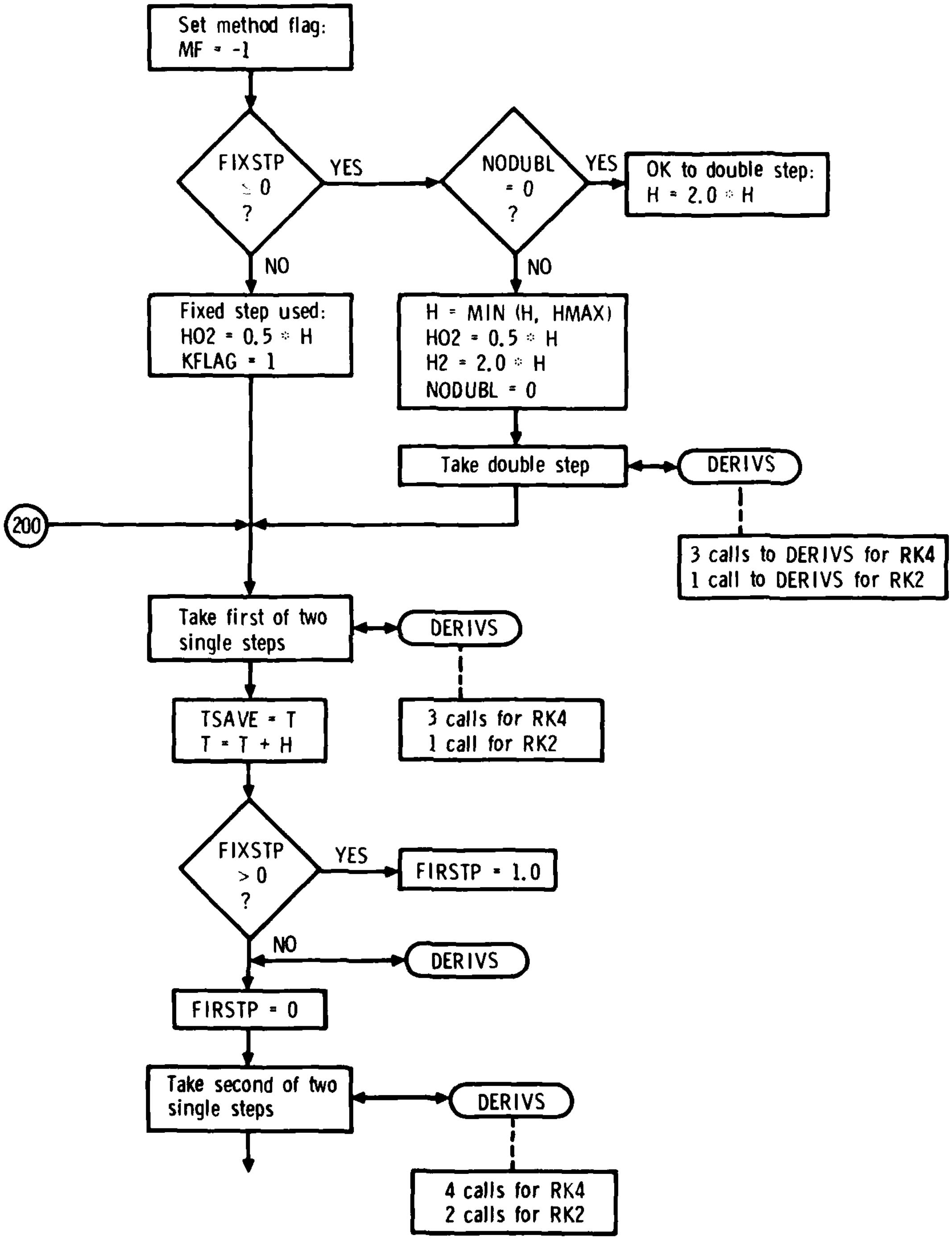


D-4-c-iv. Flowchart of Subroutine ADAMS (page 2 of 2)

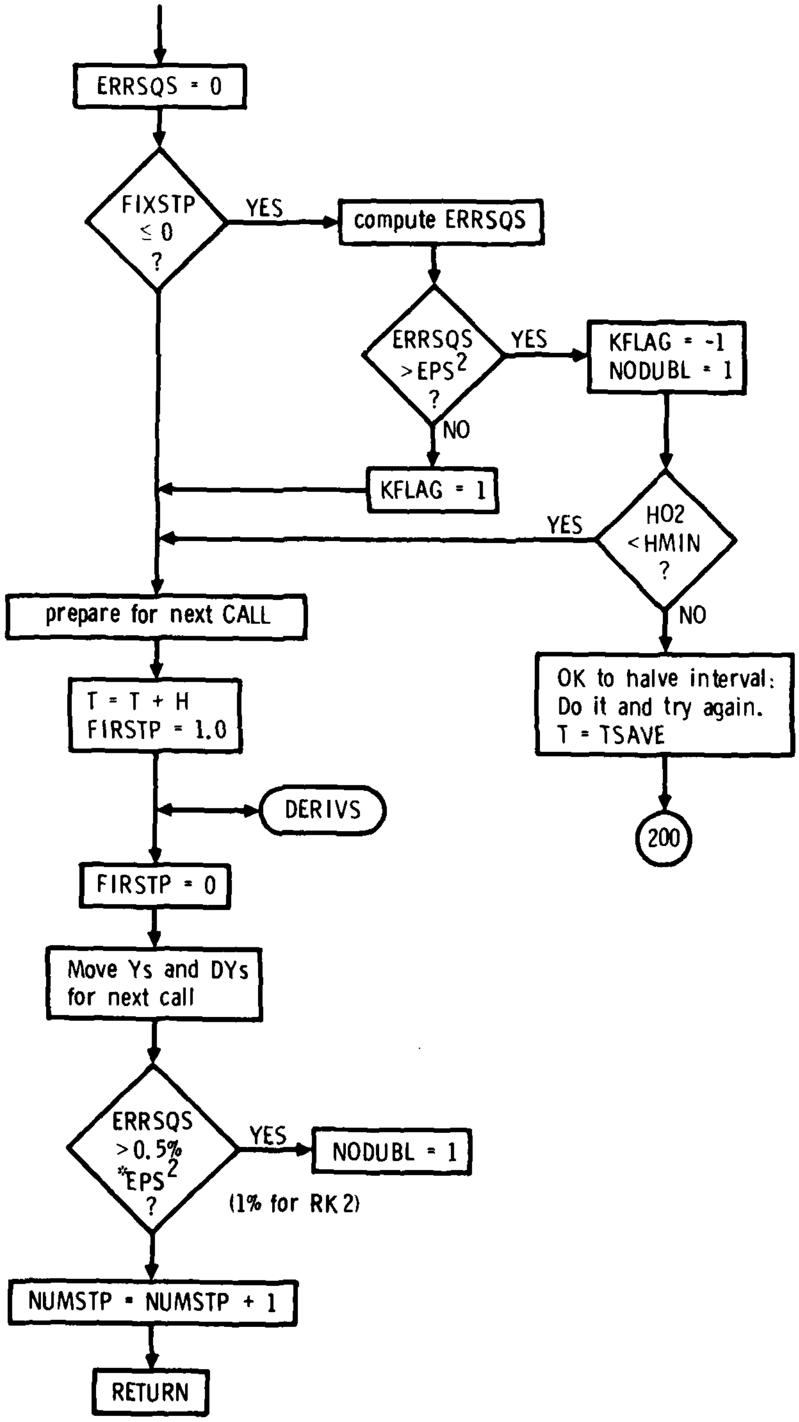


K ≥ NS ?
YES
Compute those coefficients of formulas which are changed when stepsize H is changed
NO
200
Predict a solution
XOLD = T
T = T + H
DERIVS
Evaluate derivatives at T using predicted solution.
Estimate errors at orders K, K-1, K-2
KNEW = K
Should order be lowered ?
YES
KNEW = K - 1
NO
400
Error acceptable ?
YES
Successful step:
KOLD = K
HOLD = H
Correct predicted solution
DERIVS
Evaluate derivs using corrected solution
Update differences for next step. Determine best order and step size for next step
RETURN
NO
300
Unsuccessful step:
Restore T and related variables
IFAIL = IFAIL + 1
IFAIL -3 ?
POS
Compute optimum H
NEG
Compute new H
K = KNEW
0
KNEW = 1
New H OK ?
YES
100
NO
CRASH = 0
EPS = EPS + EPS
RETURN

D-4-c-v. Flowchart of Subroutine ESPRK4 (ESPRK2) (page 1 of 2)

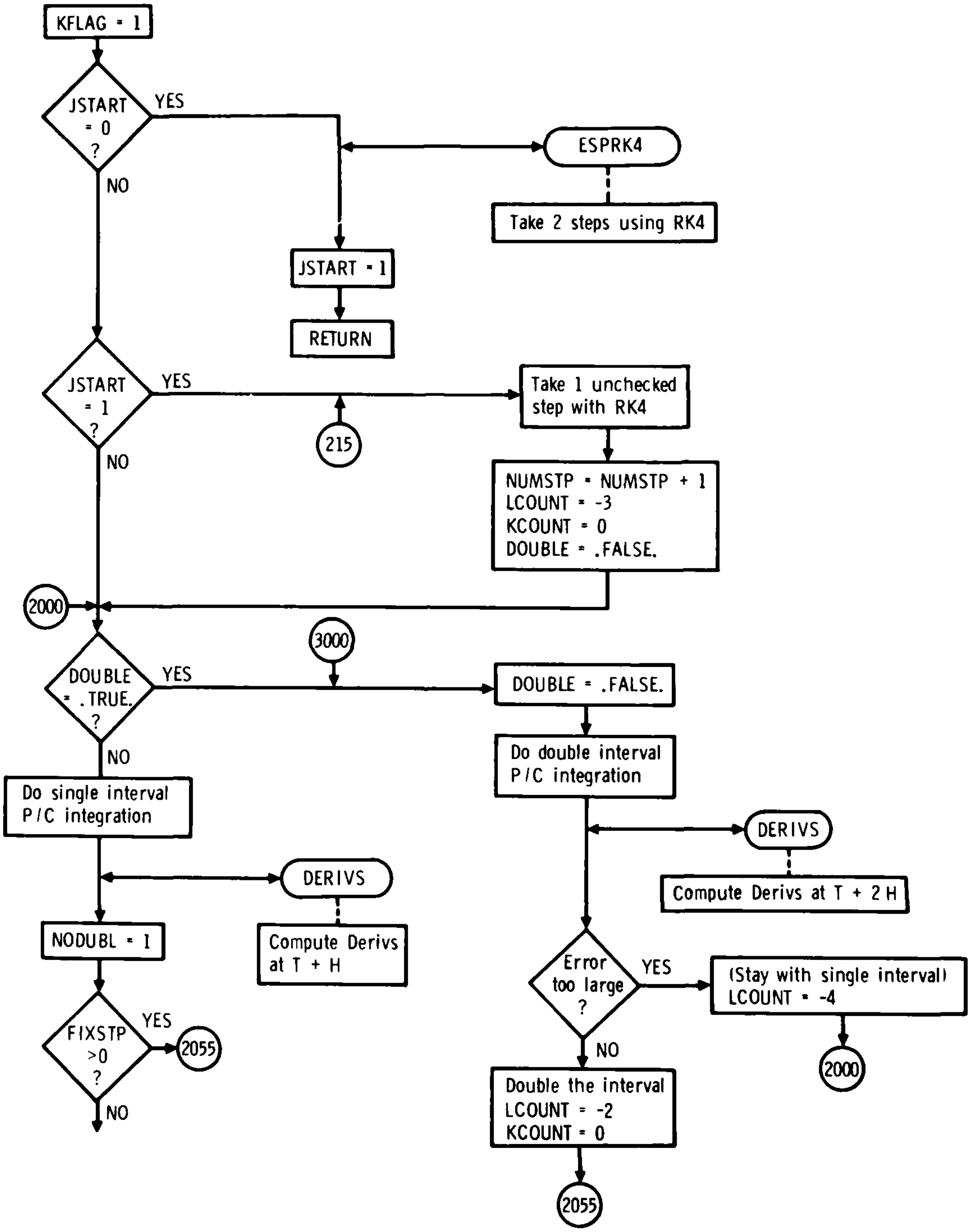


D-4-c-v. Flowchart of Subroutine ESPRK4 (ESPRK2) (page 2 of 2)

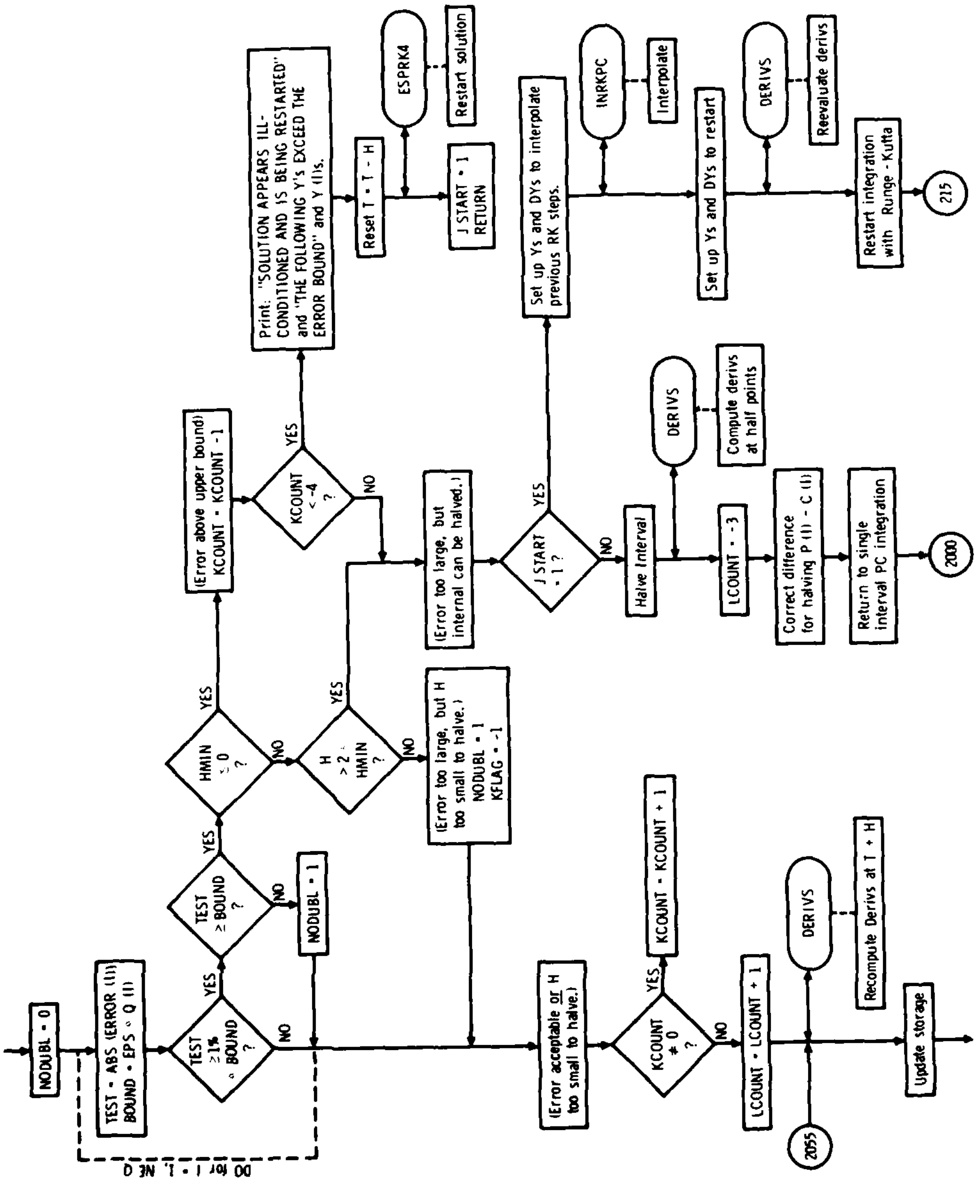


D-4-c-vi. Flowchart of Subroutine ESPPC

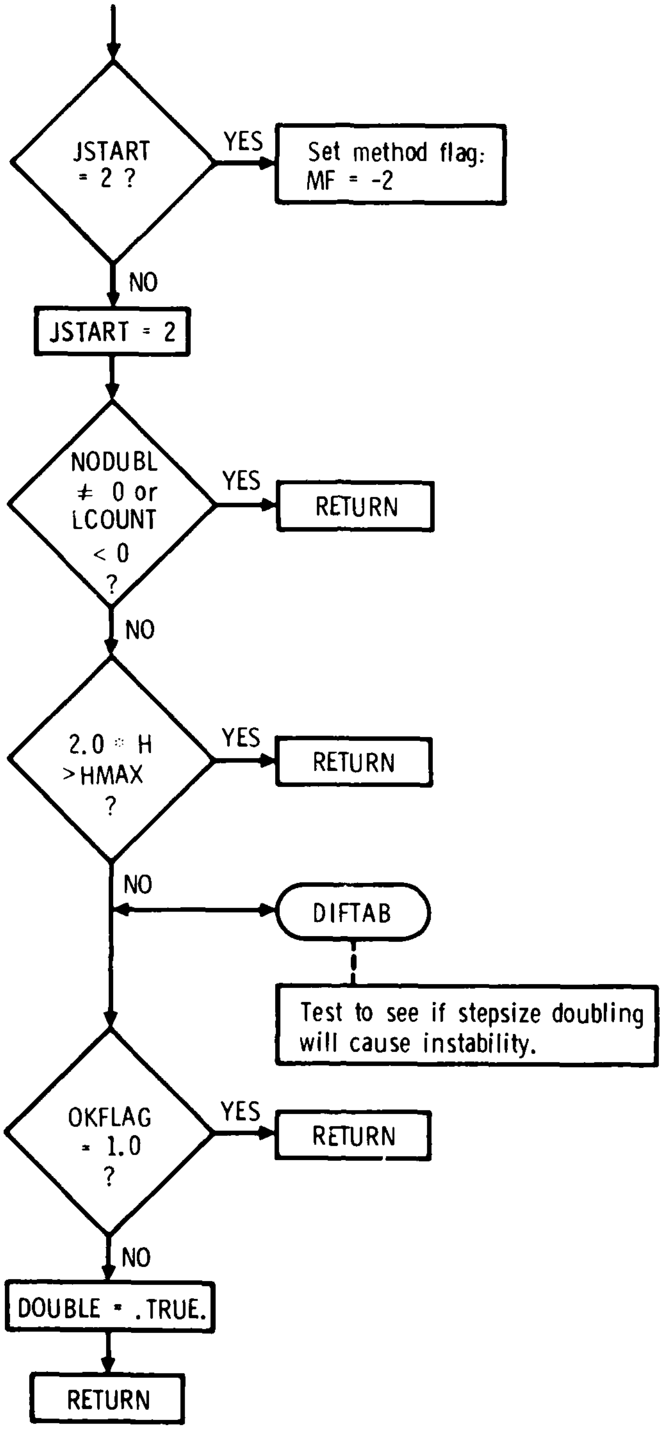
(page 1 of 3)



D-4-c-vi. Flowchart of Subroutine ESPPC (page 2 of 3)


NODUBL = 0
TEST = ABS (ERROR (I))
BOUND = EPS * Q (I)
DO for I = 1, NEQ
TEST ≥1% * BOUND ?
YES
NO
TEST ≥ BOUND ?
YES
NO
NODUBL = 1
HMIN ≤ 0 ?
YES
NO
H > 2 * HMIN ?
YES
NO
(Error too large, but H too small to halve.)
NODUBL = 1
KFLAG = -1
(Error above upper bound)
KCOUNT = KCOUNT -1
KCOUNT < -4 ?
YES
NO
Print: "SOLUTION APPEARS ILL-CONDITIONED AND IS BEING RESTARTED" and "THE FOLLOWING Y's EXCEED THE ERROR BOUND" and Y (I)s.
Reset T = T - H
ESPRK4
Restart solution
J START = 1
RETURN
(Error too large, but internal can be halved.)
J START = 1 ?
YES
NO
Set up Ys and DYs to interpolate previous RK steps.
INRKPC
Interpolate
Set up Ys and DYs to restart
DERIVS
Reevaluate derivs
Restart integration with Runge - Kutta
215
Halve Interval
DERIVS
Compute derivs at half points
LCOUNT = -3
Correct difference for halving P (I) - C (I)
Return to single interval PC integration
2000
(Error acceptable or H too small to halve.)
KCOUNT ≠ 0 ?
YES
NO
KCOUNT = KCOUNT + 1
LCOUNT = LCOUNT + 1
2055
DERIVS
Recompute Derivs at T + H
Update storage

D-4-c-vi. Flowchart of Subroutine ESPPC
(page 3 of 3)



APPENDIX E

INTEGRATOR EQUATIONS

APPENDIX E

INTEGRATOR EQUATIONS

In all descriptions given below, assume a differential equation of the form

$$\frac{dy}{dt} = f(t, y)$$

with y a vector. For ease of notation, $y_n = y(t_n)$.

E-1. ADAMS INTEGRATION

Adams integration is a highly complex variable-order, variable-step algorithm, which is completely documented in Ref. 6. As its complexity precludes condensation, the user is referred to Section IV-B and Appendix D-4-c-iv for an overview of the method and to Ref. 6 for the actual algorithm.

E-2. SECOND-ORDER RUNGE-KUTTA

$$y_{n+1} = y_n + 0.5\, h(k_0 + k_1) \tag{E-1}$$

where

$$k_0 = f(t_n, y_n)$$

$$k_1 = f(t_n + h, y_n + hk_0)$$

In the RK2 fixed-step mode, Eq. (E-1) is used to take two steps at a time before checking for such items as print, plot, and switchings. In the RK2 variable-step mode, a step of size 2h is taken first and compared with the result of two normal steps. If $\tilde{y}_{n+2}$ is the result of the 2h step and y_{n+2} is the result of 2 normal steps, the estimated error vector is

$$\text{err} = (y_{n+2} - \tilde{y}_{n+2})/3.0$$

which is added to y_{n+2} to improve the accuracy. This error vector is used to control stepsize based on the test outlined in Section IV-C, namely, let

$$\text{bnd} = \text{eps X max } (y_{n+2}, q)$$

then

1. If err > bnd in <u>any</u> component, halve the stepsize (if allowed) and retry.
2. If err < bnd/30.0 in <u>every</u> component, set h = min(stepmax, 2h) for the next step.

E-3. FOURTH-ORDER RUNGE-KUTTA

$$y_{n+1} = y_n + h(k_0 + 2k_1 + 2k_2 + k_3)/6.0 \qquad \text{(E-2)}$$

where

$$k_0 = f(t_n, y_n)$$

$$k_1 = f(t_n + 0.5h,\ y_n + 0.5hk_0)$$

$$k_2 = f(t_n + 0.5h,\ y_n + 0.5hk_1)$$

$$k_3 = f(t_n + h,\ y_n + hk_2)$$

In the RK4 fixed-step mode, Eq. (E-2) is used to take two steps at a time before checking for such items as print, plot, and switchings. In the RK4 variable-step mode, a step of size 2h is taken first and compared with the result of two normal steps. If $\tilde{y}_{n+2}$ is the result of the 2h step and y_{n+2} is the result of the two normal steps, the estimated error vector is

$$\text{err} = (y_{n+2} - \tilde{y}_{n+2})/15.0$$

which is added to y_{n+2} to improve the accuracy. This error vector is used to control stepsize based on the test outlined in Section IV-C, namely, let

$$\text{bnd} = \text{eps X max}(y_{n+2}, q)$$

then

1. If err > bnd in any component, halve the stepsize (if allowed) and retry.

2. If err < bnd/150.0 in every component set h = min (stepmax, 2h) for the next step.

E-4. HAMMING PREDICTOR CORRECTOR

Once four back values have been created using Eq. (E-2), the following formulae are used at each step (primes indicate derivatives).

$$p_{n+1} = y_{n-3} + 4h(2y'_n - y'_{n-1} + 2y'_{n-2})/3$$

$$m_{n+1} = p_{n+1} - 112(p_n - c_n)/121$$

$$c_{n+1} = [9y_n - y_{n-2} + 3h(m'_{n+1} + 2y'_n - y'_{n-1})]/8$$

$$y_{n+1} = c_{n+1} + 9(p_{n+1} - c_{n+1})/121$$

Stepsize control is based on the vectors err and bnd, where

$$\text{err} = 9(p_{n+1} - c_{n+1})/121$$

$$\text{bnd} = \text{eps} \max(y_{n+2}, q)$$

1. If err < bnd/100 in every component, attempt to double the stepsize.

2. If err > bnd is any component, halve the stepsize if allowed.

If interval halving is required, the required back values for y are created by interpolation and the derivative values by calling the derivative routine. Specifically, the formulae used for the interpolation of back values are

$$y_{n-1/2} = [45y_n + 72y_{n-1} + 11y_{n-2} + h(-9y'_n + 36y'_{n-1} + 3y'_{n-2})]/128.$$

$$y_{n-3/2} = [11y_n + 72y_{n-1} + 45y_{n-2} - h(3y'_n + 36y'_{n-1} - 9y'_{n-2})]/128.$$

The difference $p_n - c_n$ from the previous step is divided by 32 to account for halving, multiplied by 32 to account for doubling, and set to zero following an RK4 restart.

Stepsize doubling is only attempted if $err < bnd/100$ and the number of successful predictor-corrector steps has been at least

1. 3 after a RK4 restart or halving
2. 2 after a successful doubling
3. 4 after a doubling failure

APPENDIX F

SPECIAL CASES: MULTIPLE RUNS AND LARGE SIMULATIONS

APPENDIX F

SPECIAL CASES: MULTIPLE RUNS AND LARGE SIMULATIONS

F-1. MULTIPLE RUNS

F-1-a. Multiple Runs Varying Run-Time Data Cards

If a series of job runs is to be made in which the only changes from one run to the next are in items which can be input on run-time data cards, then any number of runs can be made as one job. The necessary run-time cards are simply stacked in sequence, according to the following rules:

- IVs, PARs, SWMEMDATA, Qs, EPS, and all other run time data cards except SWMEMSET retain their values until they are reset by the user's program or on a new run-time card such as *IV or *PAR.
- *SWMEMSET, if used, must be redefined for each run since it is changed during execution.
- Each *RUN card produces reexecution of the program as soon as it is encountered, so must always be the last run-time card of a case.
- *A set of *GRAPH cards must follow each *RUN from which* plots are expected.

EXAMPLE:

```
      .
      .
      .
    *ENDIC                                  |
    *IV  0.5  Y3=0.01  $                    |
    *PAR  5.0  3.14  57.6  10.0  $          |
    *SWMEMDATA                              |  First run, with plots.
    1  1.0  2.0  0.7  0.7  1.0  $           |
    *SWMEMSET  1  $                         |
    *EPS  ALL=1.0E-10  $                    |
    *RUN  3  0.  1.0  100.  $               |
    *GRAPH  1  3                            |
    *PAR  10.0  P4=20.0  $                  |  Second run: reset SWMEM in
    *SWMEMSET  1  $                         |  saturation; change PARs, all
    *RUN  3  0.0  1.0  100.  $              |  others the same; no plots.
    *IV  0.1  0.1  0.1  $                   |  Third run: reset SWMEM in
    *SWMEMSET  1  $                         |  saturation, change IVs, use
    *RUN  3  0.  1.0  100.  $               |  PARs from second run, all
    *GRAPH  2  3                               others from first, make plots.
      .
      .
```

F-1-b. Multiple Runs with the Same Run-Time Data Cards

Sometimes the only changes from run to run are in the data read from the user's file (see case below) and the user has a rather lengthy list of run-time cards (for example, lengthy *GRAPH cards) which he prefers not to duplicate for each of the stacked cases. This can be avoided in the following manner:

- Use a *RETURN card as the last run-time card, after *RUN and all *GRAPH cards.
- Have ICCØMP read the data from the user's file and terminate program execution when all data is exhausted.
- Write the MAIN program, being sure to declare the user's file on the PRØGRAM card, and include logic to backspace TAPE12 (this is the file on which run-time cards reside during execution) after each case exactly as many command cards as are needed for one case and then loop back to the call to ESP. (If all * command cards are used for each case, TAPE12 may be rewound instead of backspaced.)

F-1-c. Multiple Runs Varying Data to be Read In by User

If the user has data decks he wishes to read in from ICCØMP and he wants to stack up a number of cases, he may use the same stacking of run-time cards as shown in the above example, but must also do several other things:

- Either stack his data cards for each case immediately following the *RUN to which they apply (see Section VII-E) or place all of the data on a separate file before running the ESP job and declare this file on the PRØGRAM MAIN card by writing his own PRØGRAM MAIN.
- Read the data in from ICCØMP by means of READ or NAMELIST statements. [If he plans to test for the end of data for a given case by using IF (EØF..., he must be sure to write EØFs on his data file when he creates it, by using the FØRTRAN ENDFILE n between data sets.]

F-1-d. Making Plot Overlays of Data from Multiple Runs

Normally the tape containing the plot data is always rewound whenever a *RUN card occurs. However, by using *RUNC in place of *RUN for cases after the first, the user may cause subsequent plot information to be written onto the file following the plot data from the previous case(s). To retrieve this new information for plotting the user must specify the appropriate case number in parentheses following the *GRAPH.

EXAMPLE:

```
        :
    *MAXPLØTS  25
    *RUN  12  0  1  10  $
    *PAR  P86 = 34.5  $
    *RUNC  12  0  1  10  $
    *PAR  P72 = 12.5  $
    *RUNC  12  0  1  10  $
    *GRAPH  1  2
         PRINTER PLØT ØF DATA FRØM FIRST CASE
    *GRAPH  (2)  1  2
         PRINTER PLØT OF DATA FRØM SECØND CASE
    *GRAPH  1  3  TYPEF
         FILMPLØT ØF DATA FRØM CASES 1-3
    *GRAPH  (2)  1  3  TYPEF ØVERLAY
    *GRAPH  (3)  1  3  TYPEF ØVERLAY
```

F-1-e. Running the Solution Backward and Forward

Boundary value problems and other problems where it is desired to have the capability of running the independent variable in either direction may be handled by having one PAR, say PAR(99), be +1.0 for forward solution and be -1.0 for backward solution. Thus use

```
          T = PAR(99) * T

           | Expressions defining the |
           | derivatives in DY        |

          T = PAR(99) * T
          DØ n i=1, neq
    n     DY(i) = PAR(99) * DY(i)
```

in the derivative routine and an arrangement such as the following for the run time cards:

```
    *PAR P99 = 1.0  $                  |
    *RUN  3  2.0  0.1  10.0  $         |  forward solution
    *PAR P99 = -1.0  $                 |
    *RUN  3  -10.0  0.1  -2.0  $       |  backward solution
```

It will also be necessary to copy DY into Y0 in ICCØMP by including:

```
CØMMØN/BASIC/T0, TF, TP, Y0(100), YPRNT(100), DY(100)
```

F-2. USING ESP FOR LARGE SIMULATIONS

F-2-a. Maximum Dimensions

In general, any combination of ESP variables and special facilities may be used within one program. However, each of the following items is limited to the total number indicated:

Derivatives:	100, whether defined as DYs or in *BLØCK form.
Discontinuities:	150 total 50 defined as *SWTCH 50 defined as *SWMEM 50 defined as EVENTS
Print Variables:	60 defined by *PRINT plus any number of variables that are user-formatted.
Plot Variables:	100 if using PLØT and *GRAPH or any number if user writes his own plot file and uses other means of plotting.
Parameters:	100 stored and passed by PAR array plus any number stored and passed by user.

F-2-b. Maintaining Flexibility

Since especially large simulations often require changes and revisions, it is highly desirable to structure them in a manner which makes additions and deletions as painless as possible. Below is a suggestion for one method of maintaining flexibility in numbering and referencing the derivatives, which the user may find adaptable to his program.

The basic goal is to start with a structure which eases the problems associated with the inevitable changes required. The approach suggested here is to modularize and to use pointers such that the modules have maximum independence.

- Define a common block containing two arrays, each having at least as many words as there are modules, e.g., CØMMØN/IPØINT/IPØINT (50), NLØCAL(50)

- Set NLØCAL(I) equal to the number of derivatives defined in the Ith block.

- Set IPØINT(1) = 0 and define the remainder of the IPØINTs by

  IPØINT(I) = IPØINT (I-1) + NLØCAL (I-1)

  for I=2, . . .

- Within the Ith module (it need not be a separate subroutine) define the DYs by, say,

```
LØC = IPØINT (I)
DY (LØC + 1) = ...
DY (LØC + 2) = ...
 .
 .
 .
```

With this scheme one need only know the correspondence of physical variables within a module. Thus, if the angular displacements of body 5 were the fourth, fifth, and sixth variables within module 5, they could be used anywhere else by

```
LØC = IPØINT (5)
AD1 = Y (LØC + 4)
AD2 = Y (LØC + 5)
AD3 = Y (LØC + 6)
```

This approach may also be useful if the number of Y's varies with the input such as is encountered in structures programs.

F-2-c. Production Runs with a Compiled Program

If produc.ion runs are to be made with an ESP program which requires considerable time for PRECØMP, WHELP, and/or FTN compilation, it may be desirable to create a binary version of the program and a separate TAPE12 file so that runs can be made without recompilation. This may be done either with cards or files via the following steps:

- Using the usual control cards for an ESP (and WHELP) program, compile the source program and either punch out the LGØ file (binary) or catalog it as a permanent file.

- Add to the run-time data cards those cards normally written by PRECØMP, namely, *SWTCHES, *SWMEMCNT, and *HEADINGS, so that the run-time data section looks like this (These cards also may be used as a deck or put on a permanent file.):

  *SWTCHES n [where n is actual number of *SWTCHs used.]

  *SWMEMCNT n [where n is actual number of *SWMEMs used.]

  *HEADINGS name$_1$...name$_m$ \$ [where name$_1$...name$_m$ are actual print headings specified in *PRINT statement. If *PRINT is not used, this card is unnecessary.]

  (Any optional run-time cards, such as *PAR, *IV, etc.)

  *RUN... [usual format]

  *GRAPH... [usual format]

  *RETURN

To make runs:

- If the compiled program and run-time cards are on files, simply attach the program file and call it LGØ, attach the TAPE12 file and call it TAPE12, and then execute LGØ.

Example:

```
ATTACH(LIB1, 2NEWRESP)
ATTACH(LIB2, 3FTNPLØT LIB)
LIBRARY (LIB1, LIB2)
ATTACH(LGØ, 2BINARYPRG, ID=VAP185)
ATTACH(TAPE12, 2TAPE12, ID=VAP185)
LGØ.
```

- Alternatively, the binary card deck and the run-time cards may be used in the following setup:

```
    ATTACH(LIB1, 2NEWRESP)
    ATTACH(LIB2, 3FTNPLØTLIB)
    LIBRARY (LIB1, LIB2)
    COPYS(INPUT, LGØ)
    CØPYS(INPUT, TAPE12)
    REWIND(TAPE12)
    LGØ.
7-8-9
    [Binary deck of user's program]
7-8-9
    [Run-time cards as shown above]
6-7-8-9
```

APPENDIX G

DEBUGGING SUGGESTIONS

APPENDIX G

DEBUGGING SUGGESTIONS

G-1. TERMINATION DURING PRECØMP

Check control cards.

Check deck structure.

Look for diagnostic message at end of listing.

Check card formats for such items as required blanks and $s.

If Precompiler reads off END-ØF-FILE, check for $ terminators, especially on *PRINT card.

G-2. TERMINATION NEAR BEGINNING OF EXECUTION

Does NEQ on run card agree with highest numbered derivative?

Is every DY(i) ($i \leq$ NEQ) defined, even if just set = 0?

Check derivative equations carefully.

Check stepsize and any stepsize limits you may be specifying. [It may be helpful to print out the stepsize (H).]

If system seems to immediately go unstable, check HMIN (see Appendix G-6).

G-3. TIME LIMIT DURING STARTING PROCEDURE

Try running the program using all Runge-Kutta integration.

Reduce the print interval to obtain more printout.

Print the stepsize, H, in order to monitor its behavior.

Set HMIN > 0., which causes integration to continue by accepting Y(i)'s in spite of errors.

Check equations for a very small or 0. value of DY coupled with a Y0 which is also very small or 0. In this situation the error constraints may be unduly difficult to satisfy, and increasing the corresponding Q(i) may alleviate the problem.

G-4. EXECUTION OCCURS BUT PRINTOUT IS ZERO OR INACCURATE

Check PARs, IVs, and all other inputs as they are printed initially.

Are you attempting to pass time-dependent variables as PARs?

Have you defined all output variables in ØUTPUT, eithor by computing them there or passing them through common blocks?

Check each subroutine to make sure referenced variables are defined.

G-5. DISCONTINUITIES ARE NOT WORKING PROPERLY

Have you introduced discontinuities only as SWTCHs or SWMEMs or by using FIRSTP flag?

Are switch inputs properly defined? See section on discontinuities to review restrictions.

Are the allowable timing errors (HSW, HSWM, HSWE) suitable to your problem?

Do you have switches driving switches directly? (Review Section V-E)

G-6. "ILL-CONDITIONED SYSTEM" MESSAGE*

This generally results from one of the following:

Errors in derivative equations

Discontinuities occurring that are not at switch times

Smooth functions which suddenly change quickly with no switches occurring

Suggestions for resolving this prcblem are as follows:

Check derivative equations for coding errors.

Check allowable errors (EPS and Q) to see if they are realistic for your problem.

Consider adjusting HMIN (default = 0): An HMIN > 0 will cause acceptance of Y's in spite of errors whenever H < 2.0 * HMIN. This has the effect of forcing integration past rough spots--useful in some cases but causing instability in others.

*See Section IV-D-1.

APPENDIX H

WHELP

APPENDIX H

WHELP

H-1. INTRODUCTION: HOW WHELP WORKS

WHELP is a higher-level language preprocessor, which translates normal (mathematical) engineering equations involving scalars, vectors, and matrices into FØRTRAN statements. It consists of two parts: a preprocessor which converts the WHELP equations to FØRTRAN statements, and a library of highly efficient subroutines which perform the actual matrix operations.

WHELP offers two big advantages: speed and accuracy. It saves coding time by permitting matrix equations to be coded much in the same form as they are written (instead of as a series of subroutine calls); it helps to reduce programmer errors since the user's code is simpler than otherwise required and maintains a close resemblance to the equations it represents; and it facilitates debugging because any equation or coding errors that do crop up are easier to find.

For example, the equation

$$\boxed{A = B^T * C + D * E}$$

where A, B, C, D, and E are
3 x 3 matrices

which normally the user would have to code as a series of subroutine calls, can be coded in WHELP with the one card:

```
A = B,*C + D*E $
```

WHELP functions by first setting up a list of those variable names (scalars, vectors, and matrices) which the user has declared as WHELP variables. It then searches the user's coding until it recognizes one of those variables on the left-hand side of an equal sign. This is interpreted as the signal that the right-hand side is a WHELP (vector-matrix) expression and must ue translated to FØRTRAN. WHELP then scans the expression up to the $, comparing each variable to the list of declared WHELP variables, to ascertain which represent scalars, vectors and matrices, and to determine their dimensions. Using the proper dimensions it then interprets the operator symbols (+, *, etc.) to generate calls to the appropriate subroutines to perform matrix multiplication, addition, transposition, etc. The end result of the WHELP processor is an executable FORTRAN program, in which the user's matrix equations will appear as comment cards followed by the subroutine calls needed to implement them. When this program is then compiled and executed, the needed matrix subroutines will be loaded from the WHELP library. The user, therefore, has two tasks to perform: declaring and dimensioning those variables which will be WHELP variables and writing his vector-matrix equations in a form that WHELP can interpret.

Basic WHELP assumes vectors and matrices of fixed size, but WHELP may also be used with arrays of variable dimensions. The discussion below will start with the simplest application, fixed-dimension WHELP, and then proceed to a description of variable dimension WHELP and to some special time-saving features which have been added to WHELP.

H-2. FIXED DIMENSION WHELP

A WHELP variable is defined to be any vector or matrix which will be referenced as a vector or matrix in a WHELP expression or any scalar which is to receive the result of a WHELP computation. Any scalar or single element of an array which will b‹ referenced only on the right-hand side of a WHELP expression need not be declared as a WHELP variable.

There are two types of WHELP statements: declaration statements and WHELP equations. WHELP declaration statements always begin in column 1 with an asterisk (*) followed by the appropriate name (SAMESIZE, IDECLARE, or INFØRM: See below) and entries are terminated by a dollar sign ($) preceded by at least one blank. WHELP equations always begin with a declared WHELP variable starting in column 7 or later followed by an equal sign and the appropriate expression and are terminated by a dollar sign ($) preceded by at least one blank. Both may extend through column 72 and be continued on the next card. No continuation marks of any kind are used. The statements are assumed to continue until the $ terminator.

H-2-a. Declaring WHELP Variables

Every scalar, matrix, or vector which is to be used as a WHELP variable must appear on a declaration card within each (sub) program in which it will be so used. A declaration card is a card started in column 1 by *IDECLARE, *SAMESIZE, or *INFØRM and terminated by a $ preceded by at least one blank. Alternative forms, which are both translated into REAL statements, are

```
*IDECLARE   item   item...   item   $
```

where

item = { name
name(n)
name(n, m)

n (and m)   are integer constants specifying the number of rows (and columns) in the array

and

```
*SAMESIZE   n   m   list   $
```

where

n (and m)   are integer constants specifying the number of rows (and columns) in all arrays named in the list. The parameter n must be present, but m may be omitted, in which case all arrays named are singly subscripted.

list   is a list of names, separated by at least one blank, which are to have the dimension(s) given.

EXAMPLES:

```
*SAMESIZE   3   X   Y   Z   W   $
*IDECLARE   F(6,3)   G(12,12)   D   H(6)   $
*SAMESIZE   12   12   A   B   C   E   $
```

These will be translated by WHELP into the FØRTRAN statements:

```
REAL  X(3),   Y(3),   Z(3),   W(3)
REAL  F(6,3),   G(12,12),   D,   H(6)
REAL  A(12,12),   B(12,12),   C(12,12),   E(12,12)
```

Note that all variables will be typed real and that these are the only statements needed to dimension these variables, and in fact the variables must not be dimensioned elsewhere. Also note that blanks are the only delimiters between list items: do not insert commas. Extra blanks may be inserted between items.

Note:

1. A name is a string of 1-7 characters (1-6 characters for IBM) acceptable to FØRTRAN as a variable name.

2. Embedded blanks are not allowed in names, since blanks are used as delimiters between items, but extra blanks may be inserted between items to improve readability.

3. TEMS and CONSTS are reserved names.

4. Since both *SAMESIZE and *IDECLARE cards are translated into REAL statements, they must precede any executable statements.

5. WHELP variables must be declared in each routine in which they are to be used as such.

6. Scalar variables may be used anywhere within WHELP expressions, but if they appear on the left-hand side of a WHELP expression (for example, as the result of a dot product), they must be declared as WHELP variables.

A third form of declaration statement, *INFØRM, permits a submatrix to be treated as a WHELP variable. That is, the variable which will appear in a WHELP expression may be a subset of a vector or matrix, instead of the entire vector or matrix. *INFØRM works much like *SAMESIZE except no FØRTRAN declaration is written out. The character strings listed on the *INFØRM card are simply added to WHELP's list of "recognized" WHELP variable names, and the assumption is made that these variables are dimensioned elsewhere by *SAMESIZE, *IDECLARE, CØMMØN, REAL or DIMENSIØN statements. The format is

```
*INFØRM   n   m   list $
```

where

n (and m) — are integer constants specifying the number of rows (and columns) in all arrays named in the list. The parameter n must be present, but m may be omitted, in which case all arrays named are treated as vectors.

list = item_1 item_2 item_3...item

where

$$\text{item} = \begin{cases} \text{name} \\ \text{name(J)} \\ \text{name(1, L)} \\ \text{name(1, 1, K)} \end{cases}$$

The 1s represent a fixed location for the starting point of the subset. They could also be any other constant within the maximum array size constraints, as long as the subset referenced consists of elements that are stored contiguously in the full array.

name — must be dimensioned elsewhere within the routine and the subscripts J, L, and K denote which dimension of name is to be varied in referencing subsets of name.

Note:

1. Item is restricted to 10 characters total.
2. Items must be written without blanks since blanks separate items.
3. Only a totally contiguous subset of an array may be declared a WHELP variable in this manner. (See H-3-a, Data Storage and

Transmission.) For example, the columns of a 3 x 3 matrix BMAT could be declared on an *INFØRM card as:

```
    *INFØRM  3  1  BMAT(1, J)  $
```

but not the rows:

```
    *INFORM  1  3  BMAT(J, 1)  $
```

because the data in a matrix row is not stored contiguously. In other words, BMAT(J, 1) is the starting location for an array of the 3 next elements in storage, and since FØRTRAN always stores a matrix such as BMAT by columns, a reference to BMAT(J, 1) where J=2 would give the following:

$$\text{if BMAT} = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \quad \text{then BMAT(J, 1)} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

EXAMPLE (in WHELP code):

```
         CØMMØN/BLØCK1/B(5, 5)
         REAL MM(5, 5, 3)  A(5, 5)
 *IDECLARE   M(5, 5)   X(5)   $
 *INFØRM   1   X(J)   $
 *INFØRM   5   M(1, L)   $
 *INFØRM   5   5   MM(1, 1, K)   $
 C       STØRE THE DØT PRØDUCT ØF THE JTH CØLUMN IN X(J).
         DØ   1   J=1, 5
         L=J
         X(J)=M(1, L) . M(1, L)   $
   1     CØNTINUE
 C           ADD VECTØR X TØ THE LTH CØLUMN ØF M.
         L=2
         M(1, L)=X+M(1, L)   $
 C           STØRE M/K  AS THE KTH SUBMATRIX ØF MM.
         DØ   2   K=1, 3
         MM(1, 1, K)=M/K   $
   2     CØNTINUE
 C           USE INFØRM TØ DECLARE PREVIØUSLY DIMENSIØNED
 C           VARIABLES.
 *INFØRM   5   5   A   B   $
         B=M+IDENT(A)   $
```

Note the following points in the example above:

1. *INFØRM 1 X(J) $ tells WHELP to treat the string X(J) as a WHELP scalar. The 1 must appear in the *INFØRM statement.

2. *INFØRM 5 M(1, L) $ tells WHELP to treat the string M(1, L) as a 5-vector.

3. *INFØRM 5 5 MM(1, 1, K) $ tells WHELP to treat the string MM(1, 1, K) as a 5 x 5 matrix. The program is storing 1 5 x 5 matrix in each of the 3 planes of MM.

4. The maximum dimensions of all *INFØRM-declared variables are given elsewhere by DIMENSIØN, REAL, CØMMØN, *IDECLARE, or *SAMESIZE statements.

5. The *INFØRM card places the total character string (for example MM(1, 1, K)) in the table of recognized variables. Thereafter, the total string should be thought of as a FØRTRAN "name"; that is, its spelling is sacrosanct and no changing or substituting of variables (for example K) is allowed, and it may be referenced only by the full character string exactly as it appears on the *INFØRM card.

6. Just as any WHELP scalar which will appear on the left-hand side of a WHELP expression must be declared on an *IDECLARE card, any element of a matrix which is to be used as a scalar on the left-hand side of a WHELP expression must be declared on an *INFØRM card with dimension 1, as in the example, *INFØRM 1 X(J) $.

7. Since *INFØRM does not result in the writing of any dimension statements (and in fact will appear only as a comment card in the FØRTRAN listing of the program), it may be used anywhere within a (sub)program as long as it precedes WHELP statement references to the variables it declares.

8. *INFØRM may be very conveniently used when it is desired to declare variables which have already been dimensioned elsewhere. (Use of *SAMESIZE or *IDECLARE in the same situation, since they result in REAL statements, would produce double dimensioning, not normally allowed in FØRTRAN.)

H-2-b. Writing WHELP Expressions

A WHELP vector-matrix expression always begins with a declared WHELP variable and ends with a $, but otherwise may be written in much the same form that it is written mathematically, simply by using the symbols given in the table below for the desired matrix operations. For example, one might wish to evaluate a variable after several coordinate transformations by

$$\{XNEW\} = \alpha[A]\,[B]\,[C]\,\{XOLD\}$$

or estimate a correction term by

$$\{delx\} = [A^T A]^{-1}\, A^T \{r\}$$

Using WHELP, these are coded

```
XNEW = ALPHA* A*B*C*XØLD $
DELX = INVERSE (A, *A)*A, *R $
```

and will appear exactly like this on the first listing of the program. On the FØRTRAN file listing of the program produced by WHELP, all WHELP expressions will be rewritten as comment cards, immediately followed by the generated FØRTRAN calls (see Calling Sequences, Appendix H-2-b-ii).

Note that in the example above, the vectors XNEW, XØLD, R and DELX and the matrices A, B, and C must have been previously declared on a *SAMESIZE, *IDECLARE, or *INFØRM card.

Note also that any character string appearing in a WHELP expression which is not identical to a WHELP declared character string (ALPHA in the example above) will be treated as a scalar. That is, if XØLD is a declared WHELP vector, but the character string XØLD (I) appears in a WHELP expression, XØLD (I) is treated as a scalar. In general the acceptable string length is 1-10 characters if the first character is a letter, but is not limited for numbers. However, since a longer character string is sometimes inevitable, for example ZETA (I + 1, 2 * J), WHELP will use the correct value for ZETA (I + 1, 2 * J) but will substitute a shorter character string for the too long one when it writes out the FØRTRAN file. In any case, the total character string must not exceed 40 characters, or information will be lost.

A WHELP equation is evaluated according to the hierarchy of the operators, given in Table H-2-b-i. In expressions with like operators, evaluation occurs from left to right. However, as in standard FØRTRAN expressions, parentheses can be used to override the usual sequence of evaluation. Blanks may be used between items and operator symbols to improve readability and the expression may extend up through column 72 and continue onto the next card with no continuation marks. The end of the expression must be indicated by a dollar sign ($) preceded by at least one blank.

H-2-b-i. Table of WHELP Operators

Operation	Coding Symbol	Hierarchy	Scalars	Mixed	Vectors	Matrices
Addition	+	1	OK	▨	OK	OK
Subtraction	-	1	OK	▨	OK	OK
Multiplication	*	2	OK	OK	OK	OK
Division	/	2	OK	(Matrix/ Scalar)	▨	▨
Dot Product	•	2	▨	▨	OK	OK
Cross Product	**	3	(Exponentiation)	▨	(3x1 only)	▨
Transpose	,	3	▨	▨	OK	OK
Transpose & Multiply	,*	3	▨	▨	(A^TB)	(A^TB)
Matrix Inverse	INVERSE(A)	4	▨	▨	▨	OK
Identity Matrix	IDENT(A)	4	▨	▨	▨	OK

Blackened areas represent operations which are illegal or impossible.

Although MATRIX,*SCALAR is mathematically legal, it must be coded as SCALAR*MATRIX, or (MATRIX,)*SCALAR.

H-2-b-ii. Calling Sequences of WHELP Matrix Routines

Because it may on occasion be desirable either to call a WHELP subroutine directly or to know its calling sequence for debugging purposes, below is a list of the WHELP operators in use and the subroutine calls which will result.

Operation	Subroutine Call
C=A+B	CALL MATADD(A, NRA, NCA, B, NRB, NCB, C, NRDIMA, NRDIMB, NRDIMC)
C=A-B	CALL MATSUB(A, NRA, NCA, B, NRB, NCB, C, NRDIMA, NRDIMB, NRDIMC)
C=A*B	CALL MATMAT(A, NRA, NCA, B, NRB, NCB, C, NRDIMA, NRDIMB, NRDIMC)
$C=A^{T}*B$	CALL TRNSML(A, NRA, NCA, B, NRB, NCB, C, NRDIMA, NRDIMB, NRDIMC)
C=Scalar*B	CALL SCAMAT(SCALAR, B, NRB, NCB, C, NRDIMB, NRDIMC)
C=B/Scalar	CALL SCAMAT(1.0/SCALAR, B, NRB, NCB, C, NRDIMB, NRDIMC)
$C=A^{T}$	CALL TRNSPS(A, NRA, NCA, C)
B=0.	CALL MATZRØ(B, NRB, NCB, NRDIMB)
C=A.B	CALL TRNSML(A, NRA, NCA, B, NRB, NCB, C, NRDIMA, NRDIMB, NRDIMC)
C=A x B	CALL CRØSS(A, B, C)
B=A	CALL MØVE(A, NRA*NCA, B)
B=-A	CALL NEGATE(A, NRA*NCA, B)
A=INVERSE(A)	CALL MATINV(A, NRA, 0, 0, DET, NRDIMA)
A=IDENT(A)	CALL IDENT(NRA, A)

where: A, B, and C are WHELP arrays of conformable size for the operations indicated.

NRA and NRB are the number of rows of A and B being used.

NCA and NCB are the number of columns of A and B being used.

NRDIMA, NRDIMB, and NRDIMC are the fixed number of rows for which A, B, and C are dimensioned.

DET is the determinant of matrix A.

H-2-c. Special Features

- A=0. $ This zeros out A where A is any WHELP variable.
- The remainder of a card following a dollar sign may be used for comment. For example:

```
A = B + ERRØR   $  ADD  ERRØR  VECTØR
```

- WHELP variables may be set equal to strings of FØRTRAN expressions. The format is

$$\boxed{\text{variable} = \$\ \text{element}_1\ \$\ \text{element}_2\ \$\ldots\$\ \text{element}_n\ \$\$}$$

where

$$\text{element}_i = \left\{\begin{array}{l}\text{a constant}\\ \qquad\text{or}\\ \text{any legal FØRTRAN expression}\\ \qquad\text{or}\\ \text{*k (where k is an integer }\underline{\text{constant}}\text{ denoting}\\ \text{how many times the expression following is}\\ \text{to be repeated)}\end{array}\right.$$

EXAMPLE:

```
*IDECLARE M(2,2)  $
        .
        .
        .
        M = $ SIN(T) $ CØS(T) $ *2 $ ALPHA + R $$
produces the result:
        M(1,1) = SIN(T)
        M(2,1) = CØS(T)
        M(1,2) = ALPHA + R
        M(2,2) = ALPHA + R
```

Note:

1. Element may not be a WHELP expression.
2. Matrix variables are stored by column and therefore must be listed by column.
3. Like the FØRTRAN DATA statement, no elements may be skipped and the number of elements must not exceed the total size of the variable.
4. When used with variable dimension WHELP (see below), data will be packed into the first N*M elements of an array.

H-3. VARIABLE DIMENSION WHELP

WHELP may also be used with matrices having variable dimensions: the WHELP equations are written in exactly the same manner as they are for fixed dimension WHELP, and special forms of *SAMESIZE, *IDECLARE and *INFØRM are used to declare the variable size matrices. These special forms will be explained below, but since successful use of variable dimension WHELP depends upon an understanding of how WHELP stores and transmits data for matrices of variable dimensions, this will be discussed first. It is strongly urged that the user carefully observe the constraints on data storage imposed and implied for variable dimension WHELP and also that thorough printout and testing be done during program development.

H-3-a. Data Storage and Transmission

We are accustomed to thinking of matrices in FØRTRAN as having several dimensions, but FØRTRAN does not actually store a matrix in a two-dimensional "slot": It stores it, column by column, in a continuous string. Thus a simple two-dimensional matrix MAT(3, 3), which we represent mathematically as

$$\begin{bmatrix} 1. & 4. & 7. \\ 2. & 5. & 8. \\ 3. & 6. & 9. \end{bmatrix}$$

is in fact stored like this:

MAT(1) = 1.
MAT(2) = 2.
MAT(3) = 3.
MAT(4) = 4.
MAT(5) = 5.
MAT(6) = 6.
MAT(7) = 7.
MAT(8) = 8.
MAT(9) = 9.

As long as full matrices are used with WHELP (or FØRTRAN), the only commonly encountered implication of this is in the use of data statements to set matrix elements, where one must remember to list data by columns rather than by rows.

Furthermore, when we wish to deal with some variable size subset of a matrix, for example, if we want to use the above MAT(3, 3) as MAT(N, M) where N = 2 and M = 2, then we are accustomed to thinking of our data storage like this:

$$\text{MAT(N, M)} \quad \begin{bmatrix} 1. & 4. & 7. \\ 2. & 5. & 8. \\ 3. & 6. & 9. \end{bmatrix} \qquad \begin{aligned} \text{or MAT(N, M)} = \text{MAT(1, 1)} &= 1. \\ \text{MAT(2, 1)} &= 2. \\ \text{MAT(1, 2)} &= 4. \\ \text{MAT(2, 2)} &= 5. \end{aligned}$$

MAT(3, 3)

where MAT(N, M) occupies the first 2 x 2 positions in MAT(3, 3).

WHELP, however, assumes that the data in MAT(N, M) is stored in the first N x M locations of MAT(3, 3) as follows:

$$\text{MAT(N, M)} \quad \begin{bmatrix} 1. & 4. & 7. \\ 2. & 5. & 8. \\ 3. & 6. & 9. \end{bmatrix} \qquad \begin{aligned} \text{or MAT(N, M)} = \text{MAT(1)} &= 1. \\ \text{MAT(2)} &= 2. \\ \text{MAT(3)} &= 3. \\ \text{MAT(4)} &= 4. \end{aligned}$$

MAT(3, 3)

In other words, WHELP always assumes that the N x M elements of a variable dimension matrix are stored "packed", one immediately after the other, by columns, in the storage space allotted for the full maximum size of the array. Whether it is operating <u>on</u> a matrix or storing the results of an operation into a matrix, it will use the first N x M elements, <u>not</u> the first N rows and M columns.

Therefore, the user must always be certain that arrays to be operated on are stored "packed" and that if WHELP arrays are to be printed or otherwise used in FØRTRAN format, they must be "unpacked" by the print statement or some other means. Two subroutines are included in the WHELP library to aid the user in changing from FØRTRAN matrix format ("unpacked") to WHELP matrix format ("packed") and vice versa. They are explained below in Appendix H-3-c.

H-3-b. Declaring Variable Dimension WHELP Variables

Special forms of *SAMESIZE, *IDECLARE, and *INFØRM accomplish the task of activating variable dimension WHELP. The formats are the same as for fixed dimension WHELP except that n and m (row and column dimensions) can take either of two forms:

$$n(m) = \begin{cases} \text{integer name/integer constant} \\ \text{or} \\ \text{integer constant} \end{cases}$$

where

integer name must be 1-3 characters, beginning with a letter

integer constant is the maximum size

EXAMPLES:

```
*SAMESIZE   N/10   M/20   A   B   $
*SAMESIZE   20   M/10   C   D   E   $
*IDECLARE   A(N/10,M/20)   B(10,M/20)   C(5,5)   $
*IDECLARE   Z   X(K/30)   Y(K/20)   $
*INFØRM   N/20   M/20   FM(1,1,K)   $
*INFØRM   L/20   PT(1,J)   $
```

The following statements apply to all three declaration forms:

1. FØRTRAN subroutine calls generated by WHELP will always use the letters (if any) and result in variable-dimension computations, assuming data to be used is packed and producing packed results.

2. The numbers given as dimensions determine the maximum size of the arrays declared.

3. WHELP checks to see that the maximum dimensions of arrays are conformable for the operations indicated in an expression, but it does not check the variable dimensions to ensure they are less than the maximum dimensions nor does it check to ensure that they result in conformable matrices.

Beyond this lie some important differences in how the three statements may be used, due to the following facts:

1. FØRTRAN requires that the maximum size of an array must be stated before any subset of the array may be referenced.

2. If an array is to have variable dimensions within a subroutine, *then the integer variable names representing* those dimensions, as well as the array name, must be part of the argument list of the subroutine.

3. *SAMESIZE, *IDECLARE, and *INFØRM are all translated differently by the WHELP precompiler:

```
*SAMESIZE    N/10    M/20    A    B    $
                  produces
REAL    A(10,20),    B(10,20)
```

whereas

```
*IDECLARE    A(N/10,N/20)    B(10,M/20)    $
                  produces
REAL    A(N,M),    B(10,M)
```

and

```
*INFØRM    N/20    PS(1,J)    $
                  produces
(no declaration statement)
```

Based on these differences, some general (though by no means comprehensive) guidelines for use of *SAMESIZE, *IDECLARE, and *INFØRM may be suggested:

1. *IDECLARE may not be used in a main program. (Use *SAMESIZE.)

2. If the array name and its variable dimensions are not among the subroutines arguments, *IDECLARE may not be used to declare the array in a subroutine.

3. *SAMESIZE may be used in any routine.

4. If an array has already been dimensioned within a routine by any means, *INFØRM may be used to declare the entire array or any <u>totally contiguous</u> subset of it as a WHELP variable. Remember, though, that the data may have to be packed if it has been stored in FØRTRAN format.

5. Use of *INFØRM to declare variable dimensioned subsets of arrays is extremely error prone due to the contiguity constraint and should be used only with great care.

6.. Results should be checked carefully,. preferably on simple test data, as many possible errors will not produce any warnings, just bad results.

<u>EXAMPLE:</u>

```
      PRØGRAM TESTWH(INPUT,ØUTPUT,TAPE5=INPUT,TAPE6=ØUTPUT)
*SAMESIZE   N/5   M/5   A   B   C   $
      N = 3
      M = 3
      A = 0.   $   ZERØES ØUT FIRST N * M ELEMENTS ØF A.
      B = IDENT(B)   $   CREATES N ØRDER IDENT MATRIX, PACKED.
C STØRE DATA INTØ A IN PACKED FØRMAT
      A = $ *3 $ 1. $ *3 $ 2. $ *3 $ 3. $$
      C = A + B  $
      CALL VARDIM(A,B,C,N,M)
      CALL VARDIM2(A,B,C,N,M)
      END
      SUBRØUTINE VARDIM(A2, B2, C2, N, M)
*SAMESIZE     N/5     M/5     A2     B2     C2     $
      C2 = A2 + B2   $
      RETURN
      END
      SUBRØUTINE VARDIM2(A3, B3, C3, N, M)
*IDECLARE     A3(N/5, M/5)     B3(N/5, M/5)     C3(N/5, M/5)     $
      C3 = A3 + B3     $
      RETURN
      END
```

In this example C, C2, and C3 will all wind up with the same result, packed into the first N * M locations. Note that SUBRØUTINE VARDIM uses *SAMESIZE and SUBRØUTINE VARDIM2 uses *IDECLARE, but results are identical.

H-3-c. Packing and Unpacking Arrays

To aid the user in changing from FØRTRAN matrix format ("unpacked") to WHELP matrix format ("packed") and vice versa, two subroutines are included in the WHELP library. They are called by:

```
CALL FTØWLP(A, NRØWS, NCØLS, NDIMA)
                 and
CALL WLPTØF(A, NRØWS, NCØLS, NDIMA)
```

where

A	is the matrix to be packed (unpacked)
NRØWS	is the variable row dimension
NCØLS	is the variable column dimension
NDIMA	is the maximum number of rows for which A is dimensioned

EXAMPLES:

```
1.    *SAMESIZE   N/5   3   C   D   E   $
      C     DEFINE ELEMENTS ØF C IN FØRTRAN FØRMAT
               C(1,1) = 1.   $   C(1,2) = 4.   $   C(1,3) = 7.
               C(2,1) = 2.   $   C(2,2) = 5.   $   C(2,3) = 8.
               C(3,1) = 3.   $   C(3,2) = 6.   $   C(3,3) = 9.
               C(4,1) = 0.   $   C(4,2) = 0.   $   C(4,3) = 0.
               C(5,1) = 0.   $   C(5,2) = 0.   $   C(5,3) = 0.
               N = 3
      C     PUT C INTØ WHELP FØRMAT
               CALL FTØWLP(C,N,3,5)
      C     SET E = C(N,3) X AN N X 3 IDENTITY MATRIX
               E = C * IDENT(D)   $
```

Notes:

a. In this example the elements of C are all defined in FØRTRAN format, so before C(N, 3) can be used in a WHELP equation, *the data must be packed into the first* N x 3 *elements of* C. IDENT(D) will be a packed matrix, as will matrix E since they are the results of WHELP operations.

b. The packing operation will destroy values previously stored in C(4, 1) through C(4, 2). A subsequent unpack would leave changed values in C(4, 1), C(5, 1) and C(4, 2).

```
2.          SUBRØUTINE XYZ(A,N,M,NDIMA)
            DIMENSIØN A(NDIMA, 1)
      *INFØRM   N/20   M/3   A   $
      C     PUT INTØ WHELP FØRMAT
               CALL FTØWLP(A,N,M,NDIMA)
      C     FØRM A * A-TRANSPØSE FØR A BEING N X M
               A = A * A,   $
      C     RETURN RESULT IN FØRTRAN FØRMAT
               CALL WLPTØF(A,N,M,NDIMA)
               RETURN
               END
```

Notes:

In this example matrix A was not a variable dimension WHELP array in the calling program, so it comes to the subroutine unpacked and returns unpacked, but must be packed in order for the variable dimension WHELP equation to be executed correctly.

REFERENCES

REFERENCES

1. Hamming, R. W., "Stable Predictor-Corrector Method for Ordinary Differential Equations," JACM, Vol. 6, No. 1, January 1959.

2. Wilkinson, J. H., "Two Algorithms Based on Successive Linear Interpolation," Stanford University Computer Sciences Department, Report No. TR-CS60, April 1967.

3. Ralston, A., "A First Course in Numerical Analysis, McGraw-Hill, New York, 1965.

4. "IPD Computation Facility Computing Guide," Revised Edition, IPD Systems Programming Department, Engineering Science Operations, The Aerospace Corporation, El Segundo, California, 1 October 1974 (not available outside The Aerospace Corporation).

5. "Scope 2.1 User's Guide," Revised Edition, Control Data Cyber 70/ Model 76 Computer System 7600 Computer System, Publication No. 60372600, Control Data Corporation, Arden Hills, Minnesota, March 1978.

6. L. F. Shampine and M. K. Gordon, "Computer Solution of Ordinary Differential Equations," Freeman and Company, San Francisco, California, 1975.

INDEX

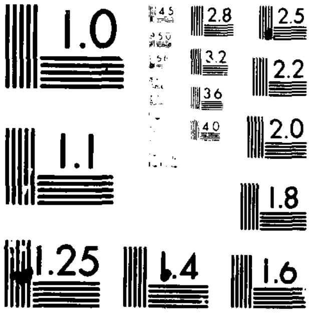
INDEX

Page

AD-A084 676   AEROSPACE CORP EL SEGUNDO CA ENGINEERING GROUP   F/G 9/2
THE ECLECTIC SIMULATOR PROGRAM (ESP) USAGE GUIDE.(U)
MAY 80 E R COFFEY, H J WERTZ   F04701-79-C-0080
UNCLASSIFIED TR-0080(9320)-1   SD -TR-80-21   NL

3 OF 3
AD
A084676

END
DATE
FILMED
6-80
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

Page

INDEX (Continued)

INDEX (Continued)

INDEX (Continued)

Page

INDEX (Continued)

INDEX (Continued)

Page

INDEX (Concluded)